NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

COMPUTER SOFTWARE PROJECT MANAGEMENT: AN INTRODUCTION

by

Samuel Matthew Liberto

June, 1991

Thesis Advisor: Donald A. Lacer

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 39	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS Program Element No. / Project No / Task No. / Work Unit Accession Number

11. TITLE (*Include Security Classification*)
Computer Software Project Management: An Introduction

12. PERSONAL AUTHOR(S) Samuel M. Liberto

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*year, month, day*)	15. PAGE COUNT
Master's Thesis	From To	June 1991	117

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (*continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUBGROUP	Computer Software Management, Software Management, Classical Software Development Life Cycle, Planning, Organizing, Directing, Controlling, CPM, PERT, COCOMO, Leadership Style, CASE, Prototyping, Risk Management, Software Engineering

19. ABSTRACT (*continue on reverse if necessary and identify by block number*)

This thesis addresses the general principles of computer software project management. The main objective is to aid perspective software project managers in dealing with the development and management of software projects. The definition of the classical software development life cycle is given. The components include system engineering, analysis, design, coding, testing, and maintenance. The thesis contains a description of the reasons why many software projects have cost overruns and late schedules. The variability of requirements and software complexity are two factors. Proper project management is one remedy to project cost overruns and late schedules. The components of software project management are planning, organizing, directing, and controlling. Many tables of comparisons and techniques for aiding software project management are given. State of the art software development techniques are discussed. Finally, a checklist to aid software managers when developing software is provided.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS REPORT [ ] DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Donald A. Lacer	22b. TELEPHONE (*Include Area code*) (408) 646-3446	22c. OFFICE SYMBOL CC/La

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited.

Computer Software Project Management: An Introduction

by

Samuel M. Liberto
Captain, United States Air Force
B.S., State University College of Buffalo 1982
M.E.A., George Washington University 1989

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(Command, Control, and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
June 1991

Author: Samuel M. Liberto
Samuel M. Liberto

Approved by: Donald A Lacer
Donald A. Lacer, Thesis Advisor

T. Hamid
Tarek Abdel-Hamid, Second Reader

Carl R. Jones, Chairman
Command, Control, and Communications Academic Group

ABSTRACT

This thesis addresses the general principles of computer software project management. The main objective is to aid perspective software project managers in dealing with the development and management of software projects. The definition of the classical software development life cycle is given. The components include system engineering, analysis, design, coding, testing, and maintenance. The thesis contains a description of the reasons why many software projects have cost overruns and late schedules. The variability of requirements and software complexity are two factors. Proper project management is one remedy to project cost overruns and late schedules. The components of software project management are planning, organizing, directing, and controlling. Many tables of comparisons and techniques for aiding software project management are given. State of the art software development techniques are discussed. Finally, a checklist to aid software managers when developing software is provided.

TABLE OF CONTENTS

I. BACKGROUND & INTRODUCTION

A. OBJECTIVE

The primary objective of this thesis is to act as an aid to program managers dealing with the development and management of computer software for the United States Air Force. The scope, or level, at which the thesis will be written is for program managers who are not computer experts or do not have much experience in the field. Therefore, the thesis will act as a "lessons learned" and a "how to paper".

B. OVERVIEW

This chapter consists of a description of the background of computer hardware and software development. A chronological history of events will show the critical discoveries and developments leading to the computer age of today. Finally, the importance and the extent of the use of software will be covered.

Chapter II will examine the inherent common problems and difficulties associated with software development in the civilian world as well in the military world. The two main

concerns researched pertain to cost and schedule over-runs. The chapter will explore why software is hard to develop and manage, and will address issues and problems that exist in software development.

Chapter III will address the classical software life cycle and military software development. Each topic will be described and explained. Next, comparisons of the two will be made.

Chapter IV will discuss software management techniques, to help allow completion of software on schedule and cost. This chapter will discuss tools that are necessary to preform project management for developing software, i.e., planning, organizing, directing, and controlling.

Chapter V will look at state of the art advances in software techniques to aid the software developer and software project manager. The advances are broken down into two major areas. The areas for improvements of software development and software project management will be described.

Chapter VI will consist of a summary section and a section on conclusions/recommendations. The summary section will re-emphasize key issues, while the conclusions/recommendations section will cover the results of this research in management of software development.

C. HISTORY OF THE COMPUTER

This section will cover a brief history of computer software and will discuss the importance of software today in the military and the civilian world. However, before software alone can be addressed, the whole or total processing system (the computer) must be covered. Finally, the importance and relationship of software to the computer will be discussed.

An appropriate definition taken from the communications standard dictionary of a computer is as follows:

> A device capable of accepting and processing data and supplying results. It usually consists of input, output, storage, arithmetic, logic, and control units. (Weik, 1983, p. 174)

In general terms, hardware, software, and firmware are the components of a computer. The hardware is the physical equipment associated with the computer (i.e., Central Processing Unit (CPU), keyboard, and the monitor). The software is the non-physical entity that directs the computer what to do (i.e., written lines of codes called programs, routines, or compilers depending on their function). The firmware is basically software written onto hardware (i.e., Electronically Programmable Read Only Memory (EPROM)). (Stevens, 1989)

The history of the computer can be traced in the following key events (Stevens, 1989):

- 1617: The invention of the sliderule.

- 1642: The mechanical calculator using wheels and gears is invented and can perform addition and subtraction.

- 1671: Improvements to the mechanical calculator are made to support multiplication and division.

- 1835: Charles Babbage invents the analytical engine, which has all the functions of a modern computer (input, output, arithmetic unit, and memory) and has a 50 digit calculation capability. Babbage is considered as the father of the computer. His girlfriend Lady Ada Lovelace is considered the mother of programming for her inputs to the analytical engine. The new software used today by the Department of Defense called ADA is named after Lady Lovelace.

- 1892: Creation of punched cards used by the US census bureau with a tabulating machine.

- 1936: Publication of a paper showing many mathematical problems solved by breaking the problem down in steps, thus illustrating that solutions can be found with instructions or a program.

- 1937: Improvements to the analytical engine using electro-mechanical components.

- 1943: The creation of the ENIAC (Electronic Numerical Integrator and Calculator) computer. This computer is made with vacuum tubes. (Wulforst, 1982, p. 63)

- 1946: Improvements to computer design are made with the publication of a paper describing how to build a computer. The principles of the paper are still used today.

- 1957: Creation of the scientific computer IBM 704 series.

- 1958-64: Replacement of tube technology with transistor technology, which launches the second generation of computers.

- 1973: Development of the hand pocket calculator and the first microprocessor.

- 1973-1978: Everyone can own a personal computer due to the mass production of microprocessors.

- 1980's: Improvements to microprocessors are made by making them faster, more powerful, and affordable.

The information contained above is illustrated graphically in Figure 1-0. The events are shown as occurrences in the graph. It can be seen that the trend for key computer technologies are increasing. The computer era has taken off tremendously in the last 50 years.

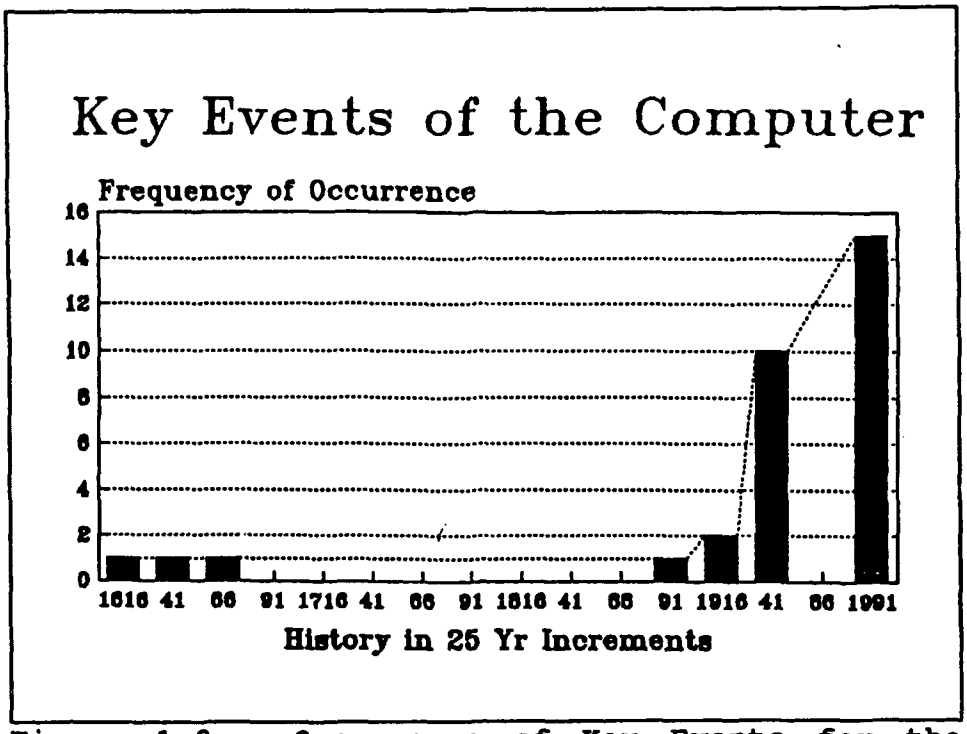

Figure 1-0. Occurrence of Key Events for the Computer.

With the advent of the first computer system in 1943 the computer age had begun. The first computer system originally, used for scientific purposes, was expanded to other applications. One of the first military applications was in 1946 when the ENIAC was commissioned by the Army's Ballistic Research Laboratory (BRL) to aid in the production of trajectory calculations. The original computers (or first generation) were made of tube technology (either vacuum or electrostatic). The second generation of transistor technology occurred in the 1960's. The third generation of integrated circuits occurred in the early 1970's. The fourth generation of the microprocessor occurred in the late 1970's.

We're in the fifth generation of very large scale monolithic integrated circuits and massive parallel processing. (Wulforst, 1982, pp. 64-70)

D. SOFTWARE BACKGROUND

The computer age also brought with it the need for software or computer programs to operate, direct and control the computer. For the original computers early software programming was nothing more than setting up the computer to perform the task using manual switches and plug in wires for operations. Efficiency in programming and software development improved as computers became more abundant and the potential was foreseen. The first types of software could be classified as microinstructions. These instructions were written in a binary form (a string of either 1's or 0's) which in turn told the computer what to do (move, store, load, add, subtract, multiply or divide) with characters/numbers. The next type of software developed could be classified as macroinstructions. These instructions were written in characters (MOV(move), STO(store), LOAD, ADD, SUB(subtract), MUL(multiply), or DIV(divide)) and the computer would have to decode what these characters meant and convert them into microinstruction and then perform the tasking. The next and final level of software can be classified as high-level languages. Most programs today are written with high-level languages. Examples of high-level languages are: FORTRAN

(FORmula TRANslation), COBOL (Common Business-Oriented Language), BASIC (Beginner's All-purpose Symbolic Instruction Code) and ADA (named after Lady Ada Lovelace and is a language using modularity for programming large-scale and real-time defense systems). High-level languages use words or characters (FORTRAN uses math characters commonly recognized and used by engineers) that are easy for the programmer to understand. The use of a compiler or interpreter is necessary to decode the high-level language to allow the computer to understand it in the microinstruction terms. (Winograd, 1986, pp. 86-90)

E. SIGNIFICANCE OF COMPUTERS AND THEIR SOFTWARE

Presently, computer systems are used in our everyday activities. The following are examples of how computers and software have touched us all in almost every aspect of our daily lives: (a) Computers run our automatic teller machines at the bank (e.g., Citibank). It takes 780,000 (780K) lines of code to support Citibank's automatic teller machines. (b) Computers are in our cars. The 1989 Lincoln continental required over 83K lines of code to run the computers that control items like the digital dashboard, brakes, air bag, engine, and suspension. (c) Grocery stores use computers at the checkout line. The IBM checkout scanner requires 90K lines of code to operate. (Schlender, 1989, p. 107)

Furthermore, computers and software are in most military weapon systems today. To show a trend one can look at the F-4 (a Vietnam Conflict era plane) which practically had no computer lines of code versus the F-16D that has 236K lines of code. The C-17 cargo plane (under development) is estimated to need between 625K to 750K lines of code compared to the older C-5A that required only 25K lines of embedded code. Finally, the Space Shuttle requires 25 million lines of code to run its computers and the B-2 needs ten million lines of code to operate its 200 computers. The costs required to develop software are staggering. The estimated cost the United States spent on software for 1989 is $112 billion, while the military spent 10% of its budget for an amount of $30 billion. Therefore, one can see the increasing importance and amounts of money being spent on software today and the upward trend for the future. (Kitfield, 1989, p. 33)

II. EXAMINATION OF SOFTWARE PROBLEMS

The development of software has practically a perfect track record. The development is usually over budget and late. The main reasons are the complexity of software development and difficulty in software management. The following section further explores the problems associated with software development and management.

A. GENERAL SOFTWARE PROBLEMS

Writing software is not an easy task. Some people view it as an art, while others think of it as a pain. All software projects have common difficulties and problems. Furthermore, there are additional problems that pertain to military systems. The following are examples of common software development difficulties and problems and will be discussed in more detail: estimating size, variability of requirements, support tools, lack of historical database, difference in personnel, hybrids, complex software, art/abstract, no reusable software, and programmers are very optimistic. (Abdel-Hamid, 1990)

1. Estimating Size

One of the hardest things to accomplish in software design is estimating the size of the project. Unfortunately, these estimates have to be done before actual work begins on the project. Why do we even care how big the project will be? It is from the estimates of the size of the project that calculations of the cost

and manpower are derived. One of the first steps in software design management is determining the size, which in turn leads to the other key specifications.

2. Variability of Requirements

Requirements and specifications of the project have to be laid out at the beginning of the project, which should be formulated from the end user's needs. The systems analyst works as the go between, transforming the user's needs into software requirements and specifications. The systems analyst interfaces with the end user and the software programmers. The problem occurs with the requirements themselves. Unfortunately, requirements change due to changes in conditions such as the environment (the situation in which the software will be used), the personnel, the hardware, or because the written requirements will not satisfy the actual user's needs.

3. Support Tools

Many support tools can help aid the manager with estimations of project size, manpower requirements, costs, and management of the project. The main problem with support tools occurs when they are not used, or are used improperly. It's obvious to see the problem of not using support tools, but if the support tools are written to work in one area of software development and the tools are applied to another area the answers could be incorrect.

4. Lack of Historical Database

One way of attempting to estimate the size, cost, and amount of effort needed for software design is through the use of existing projects. Estimating the size of the future project is done by matching information details with those of previous projects stored in a central database. Next, cost calculations can be extrapolated from the size estimations. Finally, interpolations can be made if the two projects do not match up perfectly. The problem is that most companies do not have such a database for several reasons, such as lack of resources of time, money, and people to implement, to operate and maintain the database.

5. Difference in Personnel

The number of people that work on a project depends on items like the size of the project and its importance. The people themselves vary in many ways. Examples of these variations are: skill level, programming experience (both in the language and the virtual machine being used), efficiency, knowledge, and, their individual capability. Wide variations in personnel can make estimating manhours needed to accomplish the project extremely hard.

6. Hybrids

Programming personnel can be placed in three general categories: technical, management, or a hybrid of the two. Unfortunately, most people are either the technical or management type. Therefore it is hard for a technical person to develop and

use good management techniques on the project; or it is difficult for the manager to understand all the important technical aspects. Developing and managing software takes disciplines and skills from both the technical and management fields.

7. Complex Software

Many software projects are very complex by nature. One can get an idea of the complexity by looking at the functions that must be performed, the numerous interfaces that must be made, and the accuracy in which the software must operate. Because of this complexity, many software projects are a one-of-a-kind item that can not be exactly duplicated from project to project.

8. Art/Abstract (Not Physical)

Developing software is not the same as building a bridge. The comparison is to provoke the idea that building a bridge is a science (civil engineering) while developing software is not. The civil engineer uses formulas and calculations to figure the load and stress that will be placed on the structure, while programmers on the other hand, do not have formulas or calculations to help write the software. Some refer to writing software as an art more than a science. The differences between software (languages, function, applications and development) varies enough to make development almost a new adventure each time.

9. No Reusable Software

Software is not reused very often for several reasons. One of the reasons for lack of reusable software is uniqueness. Software differs from project to project (even when using the same

language) because of its function, application, and how its developed. Another major problem encountered for reusable software is the extra quality and reliability needed. The software would need to be of such high quality and reliability, because of other people and projects depending on it, and so many varying kinds of applications. Finally, the software would need to be maintained if modifications were necessary, or mistakes were found.

10. Programmers are Very Optimistic

It seems to be human nature to not plan enough time to get a task done. People tend to be either optimistic or do not have an idea of how many things could possibly go wrong when working on any kind of project. The famous "Murphy's laws" (what can go wrong; will go wrong or things break at the worst possible moment) tend to eat up much time and energy.

B. MILITARY SOFTWARE PROBLEMS

The following are some examples of common military software development difficulties and problems. (Congress, 1989, pp. 239,240)

1. Life of Project

The software needed to support the military is different from commercial software in several ways. The life of use for the software is generally longer. It is not uncommon to have the same software in use for periods longer than twenty years. The length

of use is dependent on effectiveness compared to its commercial civilian counterpart, which is dependent upon profit and efficiency.

2. Embedded

Software is typically embedded into a weapon system. Therefore, the software has to not only be contained within the weapon system, but interface with other components (e.g., in a missile, the software interacts with the tracking, navigational, and propulsion systems). The interaction between components then leads to much stricter interface requirements compared to a commercial developed stand-alone package.

3. Real Time Critical

Many military systems have to support on going operations in real time. Software must aid the commander in decision making roles during military operations. Examples of this type of software can be categorized as command and control system software.

4. Life-Critical

One major difference between military software and civilian software is that miliary software is life-critical. Either the software is supporting our side (friendly forces) with command and control leading to better use of our assets or the software is embedded in a weapon system used to dwindle the enemy's forces.

5. Facing Intelligent Adversaries

Civilian companies may work in a *dog-eat-dog* world, where the other guy is trying to get you, in a metaphorical sense.

However, this is true in the literal sense for the military. The system has not only to cope with the harshness of the physical environment (heat, humidity, sand, etc.), but also the fact that the enemy is trying to destroy you and your system.

III. SOFTWARE LIFE CYCLE DEVELOPMENT

This chapter will describe the classic life cycle approach to software development and discuss the standard means of supporting the cycle. The life cycle exists from the software's inception to retirement and will be described in its entirety. The explanation will be in general terms to allow the reader a broad understanding of software development. Discussions will continue with general procedures employed by the military in its software development. Finally, comparisons between civilian (classic life cycle approach) and military software development procedures will be made.

A. CLASSICAL SOFTWARE LIFE CYCLE DEVELOPMENT

1. Basic Software Systems Development Principles

There are six basic principles of software system development: 1) Involve the end user throughout systems development cycle. 2) Work should be done in phases and tasks to improve management for systems development. 3) System development tasks can overlap and are not always sequential in nature. 4) An economic justification for systems development must be accomplished by treating the development as a capital investment. 5) There is no such thing as irretrievable sunken costs. Establish decision points throughout system development to determine if continuation of project is still worth while economically with respect to

potential not now much has already been sunk. 6) Documentation should be done continuously throughout system development. (Witten, Bentley & Barlow, 1989, pp. 81-85)

2. SYSTEM SOFTWARE DEVELOPMENT LIFE CYCLE

The actual phases of the system development life cycle for software are as follows: system engineering, analysis, design, code, testing, and maintenance (refer to the Figure 3-0). Improvements and newer techniques for some phases will be presented in the following chapter.

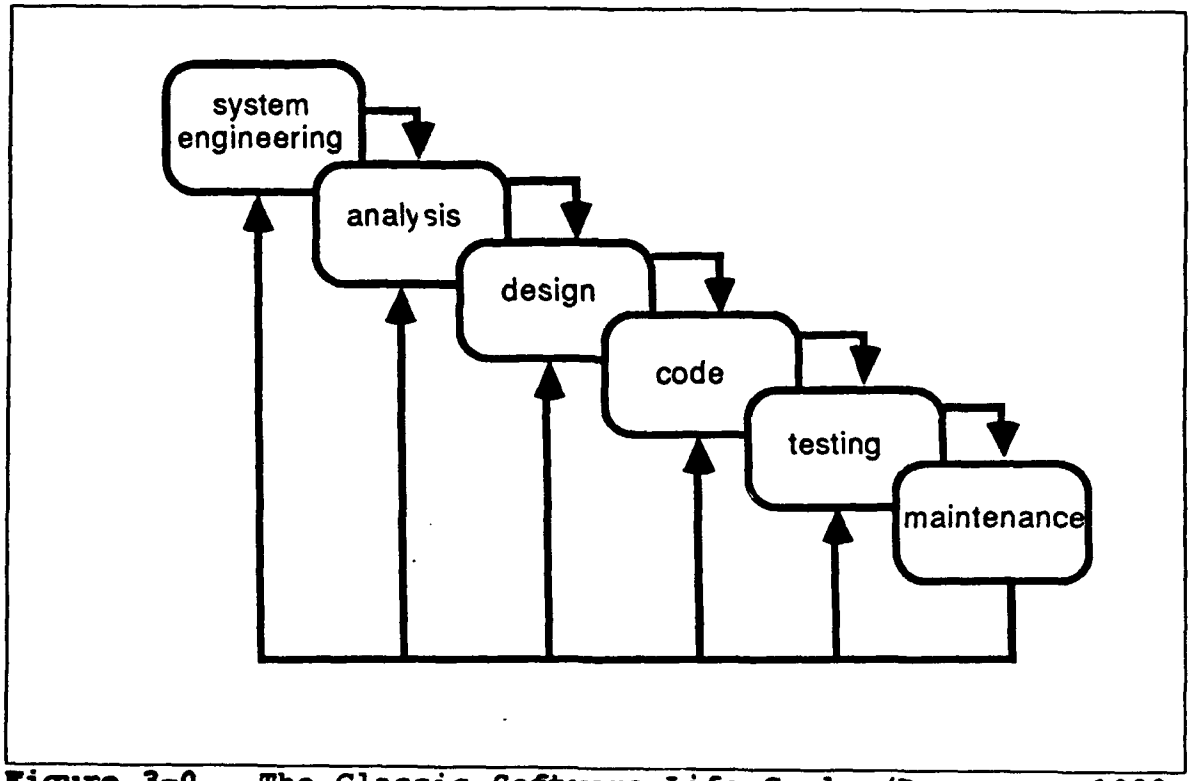

Figure 3-0. The Classic Software Life Cycle (Pressman, 1989, p. 13)

a. System Engineering

The whole existing system has to be understood and defined before any work can be accomplished. The system includes all the essential components needed for operational capability. Boundaries need to be drawn to limit the scope of the project to only the required size. The defining of and setting boundaries for the system is an example of placing the system-in-focus. Constraints may be the determining factor for boundary selection. (Jones, 1990)

The next step involves deriving the user's requirements. The software manager, and software specialist try to understand the problem being solved or the new capability being created by the software. Also, the functionality of the system is determined in this phase. "What is the system supposed to do?" one question that needs to be addressed by the software manager, and software specialist. (Ramamoorthy, Prakash, Tsai, and Usuda, 1984, p. 58)

b. Analysis

The analysis phase determines and defines the user requirements. The requirements can be used to solve an existing problem or provide a new capability. The analysis phase reviews the viable user requirements. A viable user requirement is one that is within the system resources and capabilities. Furthermore, the viable user requirements are transformed into specifications through the use of the requirement specification development process. This type of systems approach will develop software that

can be integrated into the entire system. The integration of software into the whole system increases its effectiveness. Finally, comparisons are made between the possible designs. Trade-offs must be analyzed and the best alternative chosen. Figure 3-1 illustrates how the viable user requirements are turned into specifications and passed on to the design phase. (Ramamoorthy, Prakash, Tsai, and Usuda, 1984, p. 61)

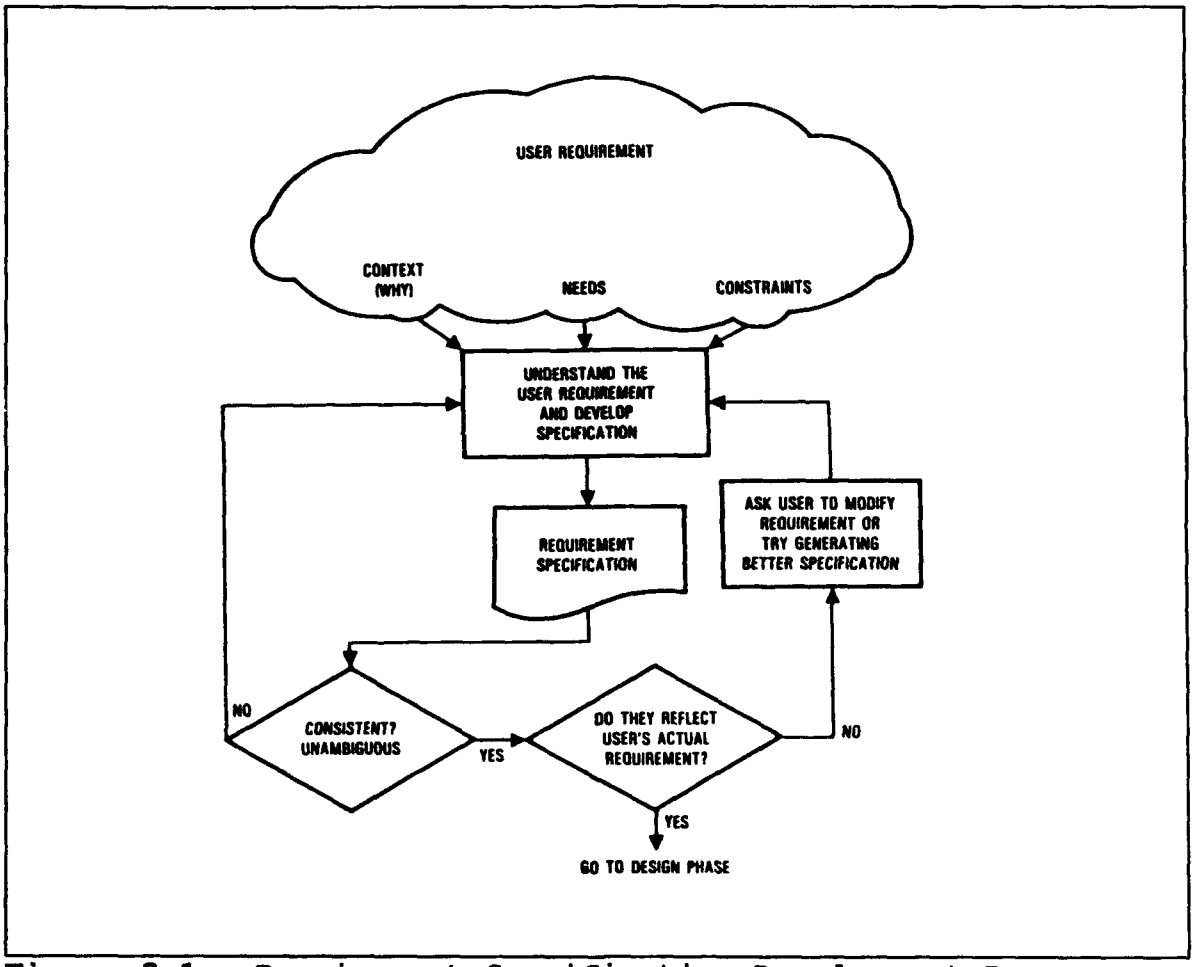

Figure 3-1. Requirement Specification Development Process

c. *System Design*

The system design phase receives the system specifications from the analysis phase. The specifications must reflect the user's requirement as shown in Figure 3-1. The design phase then breaks down the system specifications into its components. The break down continues until the components or modules are in a manageable state. This type of process is called decompositioning. Next, the decomposed modules are separated into areas by using well defined criteria. This process is called partitioning. Partitioning is done with software characteristics of minimizing complexity, providing portability, and maintainability in mind. Examples of both decomposition and partitioning are contained in Figure 3-2. (Ramamoorthy, Prakash, Tsai, and Usuda, 1984, p. 63)

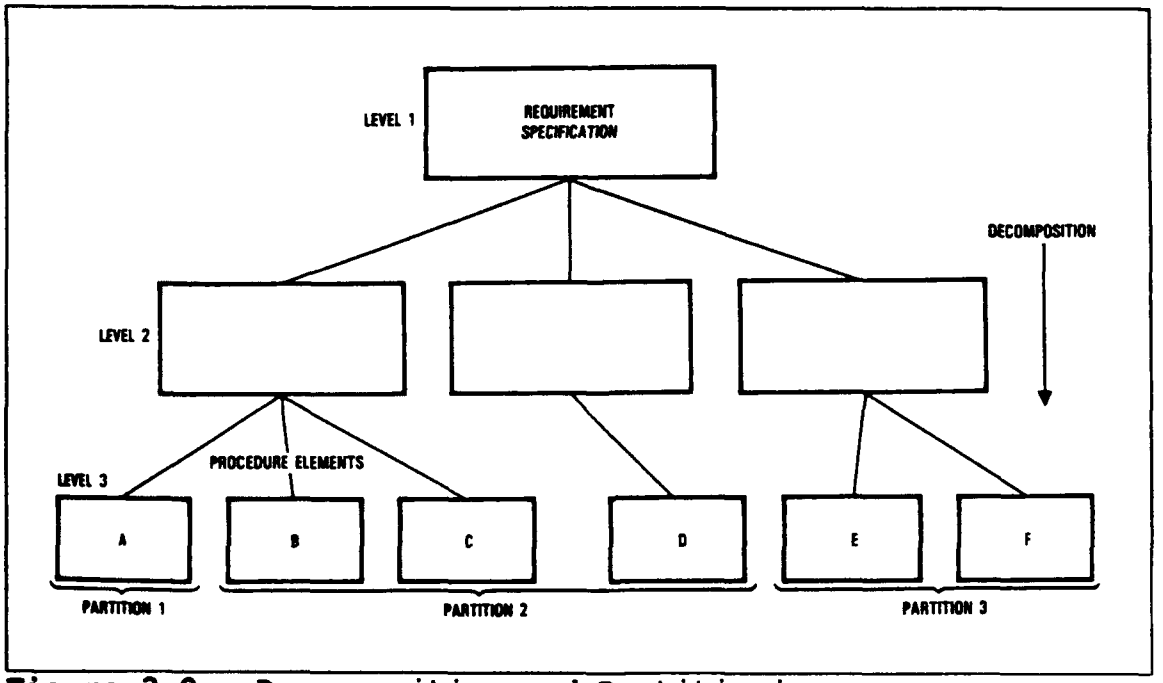

Figure 3-2. Decomposition and Partitioning

d. Code

Coding is also referred to as the programming phase. The coding phase first produces structure charts from the information produced in the partitioned modules in the design phase. Writing the code for each of these modules happens next. The detailed structure charts are used as a guide to develop the actual code (also called software) for each module. The next phase (testing) works in conjunction with this phase.

e. Test

The software has to be tested once it is written. Effective testing has to be planned to make it efficient. Four general principles of testing are:

> 1) Design test cases with the objective of uncovering errors in the software. 2) Design tests systematically; don't rely solely on intuition. 3) Establish a testing strategy that *begins* at the module level. 4) Record all testing results, and save all test cases for reapplication during software maintenance. (Pressman, 1988, pp. 193, 194)

There are three basic levels of software testing. The three levels are unit testing, integration testing and acceptance testing. Unit testing will test individual units also called modules, as mentioned earlier. Two types of unit testing are black box testing and white box testing. Black box testing has known inputs and knows the projected outputs from the module. The black box testing only cares about how the module functions (processes the inputs; giving the proper outputs) and NOT what is actually inside the box. White box testing on the other hand; IS concerned with the internal operations of the module. The white

box testing tests the actual processing of the module itself. Integration testing will take the modules that have been unit tested and start combining them. The modules are combined and tested together to ensure the system in its entirety works. Acceptance testing is the last test preformed on the system. The acceptance test is done after the installation of the system is completed. The system moves into the operation and maintenance phase, once passing the acceptance test.

f. Maintenance

Maintenance in the software development cycle means more than just fixing the undiscovered errors left over from testing. While it is part of maintenance, other parts deal with revisions and improvements that are made to the software. Improvements are sometimes called revisions and refer to an upgrade that may help the system operate better. Revisions may be necessary as changes to the environment (requirements, operating procedures, or technology). They are made to enhance the system, or are needed for the system to adapt. Finally, the users themselves may request changes to the software to provide additional benefits, and make operations easier or more efficient. Request for changes are sometimes made because the user wasn't fully aware of the systems capabilities or lack of them during the development.

B. MILITARY SOFTWARE DEVELOPMENT PROCESS

> Military software is fundamentally like advanced civilian software, only more so. That is, the properties of real-time operational software in civilian banking, airline reservations, or process control, are the same as those of

> weapon-system software. Big civilian database and file systems look essentially like military logistics, finance, and personnel software. In the operation of a ship or a base, one finds many small computers whose tasks are essentially the same as those in civilian businesses. (Report of the Defense Science Board, 1987, pp. 6,7)

1. Department of Defense Standard 2167A (DOD-STD-2167A)

The DOD-STD-2167A is the overall guiding document used for military and the remaining portions of the Department of Defense for software development.

> The purpose of this standard is to establish requirements to be applied during the acquisition, development, or support of software systems. (DOD-STD-2167A, 1988, p. 1)

The following are the major activities that occur during military software development: System requirements analysis/design, software requirements analysis, preliminary design, detailed design, Coding and Computer Software Component (CSC) testing, CSC integration and testing, Computer Software Configuration Item (CSCI) testing, and system integration & testing.

2. Comparison of Classical Software Life Cycle Development Versus Military Software Development

The relationship of military software development to the classical software life cycle development is easy to see. The major military activities are a sub-set of the life cycle. All the phases for military software development are contained within the classical software life cycle except for exclusion of the maintenance phase. The maintenance phase is left out of the major military activities because the military standard is attentive to

software development and testing. The following Table 3-0 matches the classical software life cycle to its equivalent counterpart under the DOD software development:

TABLE 3-0. COMPARISON OF THE CLASSICAL SOFTWARE LIFE CYCLE TO DOD-STD-2167A SOFTWARE DEVELOPMENT

CLASSICAL SOFTWARE LIFE CYCLE	DOD-STD-2167A SOFTWARE DEVELOPMENT
System Engineering	System Requirements Analysis and System Design
Analysis	Software Requirements Analysis
System Design	Preliminary Design, and Detailed Design
Code	Coding and CSU Testing
Test	CSU Testing, CSC integration and Testing, CSCI Testing, and System Integration and Testing
Maintenance	Not addressed because document covers software development only

The system engineering phase in the classical software life cycle corresponds with two major activities of system requirements analysis/design in the military software development cycle. The system requirements analysis/design is the equivalent to the system engineering phase in the software life cycle. The development of overall system requirements occur in this major activity in the military software development cycle.

The next phase of the classical life cycle is the analysis phase which corresponds to the software requirements analysis phase for the military version. The analysis phase

transforms the system requirements into specifications of the detailed user requirements. These specifications are sent to the design phase.

The third phase in the classical software life cycle is the same as in the military. The life cycle refers to it as the design phase while the military software development refers to it as two activities of preliminary and detailed design. Details of the user specifications are extracted by the activity of decompositioning and organized by partitioning to allow the code to be written next.

The remaining activities of the military software development are contained in the fourth (code) and fifth (testing) phases of the classical software life cycle. The military activity of coding and Computer Software Unit (CSU) testing, basically develops the code for an individual unit/module and tests it separately. The next step in the military (Computer Software Components Integration and testing (CSCI)) is to start integrating each of the computer software components and ensuring they work as a group. Then the military activity calls for testing the computer software configuration items individually to ensure they work under the Computer Software Component Integration (CSCI) testing. Finally, all software components and configuration items are integrated together in one large system, and tested under the system integration and testing.

The main difference between civilian software and military is documentation. The military requires much more

documentation than its civilian counterpart. The DOD-STD-2167A requires various documentation (referred to as deliverable products in the document) to coincide with the end of each software development phase.

> As with civilian government contracts, one of the distinguishing attributes of military contract programming is documentation. However, in the United States at least, the documentation is controlled in considerable detail by military specifications, or 'MilSpecs' as they are called. Military specifications on software documentation are so precise, so exacting, and so necessary in order to complete the contracts that a whole generation of specialists has grown up who earn their livings by documenting systems in accordance with MilSpecs. (Jones, 1986, p. 124)

Table 3-1 shows the documentation/deliverable products required for each phase of software development called out by DOD-STD-2167A. (DOD-STD-2167A, 1988, pp. 12, 13)

TABLE 3-1. DOCUMENTATION REQUIRED FOR DOD-STD-2167A

MILITARY SOFTWARE DEVELOPMENT PHASE FROM DOD-STD-2167A	DOCUMENTATION/DELIVERABLE PRODUCTS
System requirements analysis	Preliminary system specification
System design	System specification, system segment design document, preliminary software requirements specification(s), preliminary interface requirements specification, software development plan
Software requirements analysis	Software requirements specification(s), interface requirements specification
Preliminary design	Software design document(s) (pre-design), software test plan (test IDs), preliminary interface design document
Detailed design	Software design document(s) (detailed design), software test description(s) (cases), interface design document
Coding and CSU testing	Source code listings, source code
CSC integration and testing	Software test description(s) (procedures)
CSCI testing	Updated source code, software test report(s), operation and support documents
System integration and testing	Version description document(s), software product specification(s)

However, the recurring theme that needs to be stressed is that the military software development process follows the classic software life cycle. It then becomes apparent, that what applies to the classic software life cycle, also applies to military software development when addressing software management. The only major difference is the amount of documentation required for military software development. Therefore when the following

chapters address software management and new software techniques to improve software management, it applies to both the classic life cycle and to the military development.

IV. SOFTWARE PROJECT MANAGEMENT

A. INTRODUCTION

Software project management uses the same basic components that are applied in regular management. The management process is composed of four elements: planning, organizing, directing, and controlling. Planning is a formal process of making decisions, which depends upon and affects the future. Some plans are designed for the short term (weekly, monthly, or quarterly), while others are long range plans (yearly, 5-year, or 10-year). Organizing is the process of prescribing formal relationships among people and resources to accomplish organizational goals. Organizing is needed once an organization grows beyond a single member. People are grouped into labor categories, and formal organizational structures must be developed. Individual workers must be placed into individual jobs and job classifications. Directing is the element that makes things happen. This phase is sometimes known as the initiating element. It is the process where the manager can directly and personally influence the behavior of the workers. Controlling is the process the manager uses to measure the progress of the program, and compare it to the plan; to verify if any corrective measures are needed. The controlling process will then monitor the results (obtain feedback) of the corrective action(s)

to record if any further action is needed. The functions for each of the elements of management can be seen in Figure 4-0. (Newman, Warren, and Kirby, 1982, pp. 4, 40, 204, 314, and 465)

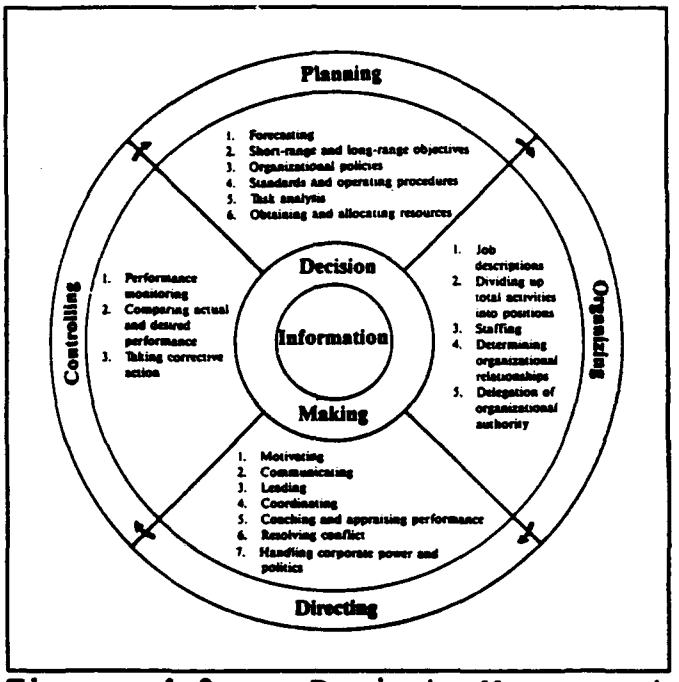

Figure 4-0. Project Management Cycle (Donohue, 1985, p. 219)

Contained in the inner center section of Figure 4-0 is information which must be processed to allow the manager to make his/her decision. The figure illustrates how the four management elements support the outer center section of decision making. The cycle of project management starts with planning and works its way around the figure in a clockwise fashion as the arrows indicate. The main functions for each element are also listed. The major functions for planning are: forecasting, short-range and long range objectives, organizational policies, standards and operating procedures, task analysis, and obtaining and allocating resources.

The major functions for organizing are: job description, dividing up total activities into positions, staffing, determining organizational relationships, and delegation of organizational authority. The major functions for directing are: motivating, communicating, leading, coordinating, coaching and appraising performance, resolving conflict, and handling corporate power and politics. The major functions for controlling are: performance monitoring, comparing actual and desired performance, and taking corrective action. The following sections will apply these four elements of management to the specific tasks necessary for successful software project management (Thayer, 1988, p. 17).

B. PLANNING

Some difficulties in software management planning are summarized as follows:

- Writing software requirements is difficult
- Planning is often incomplete or not accomplished
- Prediction of software costs and schedules are inaccurate
- Selection criteria for best procedures is not thought out completely (Thayer, 1988, pp. 23-25)

Planning is a very important aspect for software management and if done properly, can help prevent problems or difficulties for the software project down the road. Planning activities and tasks are created to help relieve the difficulties and problems associated with the major issues above. The detailed steps and activities for planning are: (Thayer, 1988, pp. 23-25)

1. Set Objectives and Goals

Objectives and goals are very specific ideas towards the formulation of software project planning. The project manager must determine what the software must achieve (the objectives) to be considered successful. The goals are a more detailed means of reaching the overall objectives of the software project. In this first step, not only what the software project must accomplish but also when and what resources are necessary must be considered.

2. Develop Strategies

Strategies can be referred to as strategic goals, because they are simply long range goals. The program manager must look at the long range goal of the organization and compare it to the long range goal of the software project. An attempt to have no conflicts between the two should be made. Finally, the software project manager must have a long range plan of meeting the goals and objectives developed in the previous section.

3. Develop Policies

A definition for a policy is as follows:

> General statements or understandings which guide decision making and activities. Policies limit the freedoms in making decisions but allow for some discretion. (Thayer, 1988, p. 23)

Policies are predetermined management decisions that can be applied in situations that meet the specific conditions identified for that particular policy. The policies act as an aid or guide for individuals to follow when routine decisions fall within the constraints.

4. Determine Courses of Action

Software project management has three general variables of time, money, and quality which help in determining his/her course of action. There is more than one way to achieve the same expenditure of money on a project, if cost were the only consideration. The project manager has various ways to achieve what is considered to be a successful project. These various ways are the courses of action that must be determined and the next phase (make decision) is used to make the best choice.

5. Scheduling

The software project manager along with inputs or considerations from the end user of the software are used to make the decision between the alternatives. The decision should involve comparisons of costs, schedule, design strategies, and risks associated with each choice. Some examples of the project management tools for scheduling are: milestone chart, Gantt chart, full wall scheduling, Critical Path Method (CPM), and Program Evaluation and Review Technique (PERT). (Donohue, 1985, p. 222)

a. Milestone Chart

The milestone chart is one of the simplest scheduling tools. It shows through the use of geometric symbols (circles, squares, or triangles) the activities needed for the completion of the software project. However, it does not show any interrelationships between the activities. Furthermore, the milestone chart shows only when activities are completed. It does not prove to be very useful for the software project manager in

terms of feedback on the progression of an activity. Therefore, this type of scheduling technique is used for small software projects or as a way to summarize complex schedules containing many tasks.

b. Gantt Chart

Another name for a Gantt chart is a bar chart. This scheduling technique overcomes many of the limitations of the milestone chart. The Gantt chart is good for small software projects (less then 25 activities) and does show the start and completion dates of the activities. The Gantt chart also reveals the progress of the activities while being administered. While it is not impossible to show interdependence among project activities with the Gantt chart, it is easier with CPM and PERT scheduling network techniques. The Gantt chart, however, does show possible overlapping of activities much easier than CPM or PERT. Therefore, it is not uncommon for the users of CPM and PERT to also use a Gantt chart by translating their scheduling network technique into a Gantt calendar chart. The main reasons are to take advantage of the Gantt chart's overlapping display capability and also because the Gantt chart is one of the easiest scheduling techniques to understand visually.

c. Full Wall Scheduling

The name full wall scheduling gives the reader a decent visualization of this scheduling technique. A wall is used to place a long roll of paper with vertical and horizontal lines to start the technique. The vertical lines represent time in weeks

and the horizontal lines represent the employees who are working on the software project. This method works best with medium sized software projects of 25 to 100 tasks and team members from three to ten. The team members must meet regularly and place marks on the wall schedule showing the progress each has made. The full wall scheduling technique works well in the area of team member interaction, due to the frequent meetings, which can clear up problems early and on the spot. The major problem with this technique however, is that it does not show the interrelationships of tasks.

d. CPM and PERT

The next two techniques of CPM (Critical Path Method) and PERT (Program Evaluation and Review Technique) can be classified as precedence networks. Both of these techniques can determine the fastest way of completing a software project or the critical path. The critical path allows the software program manager to monitor the progress of the project. The critical path is important because it must remain on time or else the whole project will be late. These two techniques are dynamic. Updates and changes can be made at any time during the project, producing a new critical path. The precedence network methods will show interrelationships between activities. However, they do not show the progress during the activities (as previously mentioned, this is when software managers who use CPM or PERT like to use the Gantt chart). For monitoring purposes the activity must be completed without interruption using the precedence network methods.

The basic difference between the two techniques is that CPM emphasizes activities, and PERT emphasizes events. Activities are the actions taken to complete the work, while an event does not occur until the work has been completed and is considered a milestone. A second difference is the way in which the two techniques determine the time estimates. In using CPM, time to complete an activity must be well known and only one number is used. PERT, however, has the capability for using probabilities of the time to complete an event. The PERT method also allows the software project manager to place three time estimate ratings of best, worst, and most likely times. The probabilities for each of the estimated times are weighted accordingly, from which is most probable to occur to the least likely to occur. Therefore, PERT is used when the specific times are not known. (Markland and Sweigart, 1987, pp. 438-440)

Each of the five scheduling techniques can be used in project software development depending on project characteristics. It is up to the software project manager to decide which technique he/she should use. The questions the software project manager wants to answer before deciding are: (1) What are the strengths and weaknesses of each software project scheduling technique; and (2) Given certain criteria, which software project scheduling technique should I choose? The following Tables (4-0, 4-1, 4-2, 4-3, 4-4, and 4-5) should help in answering those questions.

TABLE 4-0. MILESTONE TECHNIQUE (Donohue, 1985, p. 225)

CRITERIA	STRENGTHS	WEAKNESSES
APPLICABILITY	Only small errors in measurement are likely to occur if activity durations are short.	No explicit technique for depicting interrelationships.
RELIABILITY	Simplicity of system affords some reliability	Frequently unreliable because change over time. Numerous estimates in a large project, each with some unreliability, may lead to errors in judging status.
IMPLEMENTATION	Easiest of all systems because it is well understood.	Difficult to implement for the control of operations where time standards do not ordinarily exist and must be developed.
SIMULATION CAPABILITIES	_____	No significant capability.
UPDATING STATUS	Easy to update periodically. Not necessary to use computer.	_____
FLEXIBILITY	_____	Poor accommodation of frequent logic changes.
COST	Data gathering, processing and display relatively inexpensive.	The chart tends to be inflexible. Program changes require new charts.

TABLE 4-1. GANTT TECHNIQUE (Donohue, 1985, p. 227)

CRITERIA	STRENGTHS	WEAKNESSES
APPLICABILITY	Only small errors in measurement are likely to occur if activity durations are short.	No explicit technique for depicting interrelationships.
RELIABILITY	Single duration estimate for each activity avoids error due to over-complexity.	Frequently unreliable because judgement of estimator may change over time. Numerous estimates in a large project, each with some unreliability, may lead to errors in judging status.
IMPLEMENTATION	Easiest of all systems in some respects because it is well understood.	Quite difficult to implement for the control of operations where time standards do not ordinarily exist and must be developed.
SIMULATION CAPABILITIES	_____	No significant capability.
UPDATING STATUS	Easy to update graphs periodically if no major program changes. Not necessary to use computer.	May have to redo graphs because of inability to update current charts.
FLEXIBILITY	Can also be used for estimating resource requirements.	If significant logic changes occur frequently, numerous charts must be completely reconstructed.
COST	Data gathering and processing relatively inexpensive. Display can be inexpensive if existing graphs can be updated and if inexpensive materials are used.	The graph tends to be inflexible. Program changes require new graphs, which are time consuming and costly. Expensive display devices are frequently used.

TABLE 4-2. FULL WALL TECHNIQUE (Donohue, 1985, p. 226)

CRITERIA	STRENGTHS	WEAKNESSES
APPLICABILITY	Accurately depicts work sequence.	No explicit representation of activity interrelationships. Can be easily computerized.
RELIABILITY	Single duration estimate for each activity avoids errors due to over-complexity. Input from project team members often eliminates errors and problems at the outset.	Numerous estimates in a large project, each with some unreliability, may lead to significant errors in judging overall project status.
IMPLEMENTATION	Graphic display of work sequence and early discussion of project is desired by project managers. Easily explained and understood.	Time requirements and logistics problems are difficult to overcome.
SIMULATION CAPABILITIES	_____	No significant capability
UPDATING STATUS	Moderate capability. Activities are clearly identified and time estimates can be obtained as needed.	Usually requires redrawing schedule. Often difficult to update because activity interrelationships are not explicitly shown.
FLEXIBILITY	Schedule can be changed to reflect scope changes. Can be used to estimate resource requirements.	Schedules for even moderately complex projects become complicated.
COST	Can reduce overall project costs through better planning and control.	More man-hours are required than in any other system; hence this approach is often the most costly.

TABLE 4-3. CPM TECHNIQUE (Donohue, 1985, p. 226)

CRITERIA	STRENGTHS	WEAKNESSES
APPLICABILITY	Accurately depicts work sequence and interrelationships among activities.	No formula is provided to estimate probable time to completion; consequently, the technique is as valid as the estimator. The margin of error is generally less on projects with little uncertainty.
RELIABILITY	Single duration estimate for each activity avoids errors due to over-complexity.	Numerous estimates in a large project, each with some unreliability, may lead to significant errors in judging overall project status.
IMPLEMENTATION	Graphic display of work sequence and activity interrelationships is desired by managers of complex projects.	Relatively difficult to explain to those unfamiliar to approach. Complexity of schedule may intimate clients.
SIMULATION CAPABILITIES	Excellent for simulating alternative plans if computerized, especially when coupled with time-cost-resource aspects.	Requires computer for all but very small projects.
UPDATING STATUS	Good capability. Activities are clearly identified and time estimates can be obtained as needed.	Schedules for even moderately complex projects require use of computer.
FLEXIBILITY	Portions of the network can be easily changed to reflect scope changes if computerized. Can be used to estimate resource requirements if plotted on time scale.	Schedules for even moderately complex projects require use of computer.
COST	Can reduce overall project costs significantly through better planning and control.	Considerable data are required to use CPM as both a planning and status reporting tool and a computer is almost invariably required. Therefore, the cost outlay can be fairly extensive.

TABLE 4-4. PERT TECHNIQUE (Donohue, 1985, p. 228)

CRITERIA	STRENGTHS	WEAKNESSES
APPLICABILITY	PERT, like CPM is capable of depicting work sequence. The use of three time estimates should make it more accurate than any other technique.	Overly complex for small projects.
RELIABILITY	Probabilistic duration estimates may be more accurate than single estimate.	Securing three duration estimates for each activity requires more information which could introduce additional error.
IMPLEMENTATION	Graphic display of sequence and event interrelationships is desired by managers of complex projects.	The complete PERT system is quite complex, and therefore, difficult to implement. May intimidate first time users and clients.
SIMULATION CAPABILITIES	Excellent for simulating alternative plans if computerized, especially when coupled with time-cost-resource aspects.	Requires computer for all but very small projects.
UPDATING STATUS	Events are clearly identified and elapsed times can be obtained as needed.	Estimation of activity durations is quite time consuming, and calculation of expected times requires use of a computer.
FLEXIBILITY	As the project changes over time, the network and new time estimates can be readily adjusted to reflect changes. Can be used to estimate resource requirements if plotted on time scale.	Schedules for even moderately complex projects require use of computer.
COST	Can reduce overall project costs significantly through better planning and control.	More data and more computation are required than in any other system; hence the system is very costly.

TABLE 4-5. SELECTING A SCHEDULING TECHNIQUE
(Donohue, 1985, p.228)

CRITERIA	MILESTONE	GANTT	FULL WALL	CPM	PERT
Activities vs Events Oriented	Event	Activity	Event	Activity	Event
Suitability for Large Projects	Poor	Poor	Fair	Excellent	Excellent
Suitability for Small Projects	Good	Good	Poor	Poor	Poor
Degree of Control	Very Low	Low	Moderate	High	Highest
Acceptance by Users	Best	Excellent	Good	Fair	Poor
Ease of Assembly	Easiest	Easy	Hardest	Hard	Harder
Degree of Flexibility	Lowest	Low	Moderate	High	Highest
Ease of Manual Calculation	Easiest	Easy	Moderate	Hard	Hardest
Accuracy of Projections	Fair	Fair	High	Higher	Highest
Cost to Prepare and Maintain	Lowest	Low	Highest	High	Higher
Vague Project Scope	Poorest	Poor	Fair	Good	Excellent
Complex Project Logic	Poorest	Poor	Better	Excellent	Excellent
Critical Completion Date	Fair	Fair	Good	Good	Excellent
Frequent Progress Check Required	Good	Good	Good	Fair	Hard
Frequent Updating Required	Easiest	Easy	Hardest	Hard	Harder
Frequent Logic Changes Required	Poor	Poor	Poor	Fair	Fair
Appeal to Client	Good	Good	Excellent	Excellent	Excellent

6. Cost Estimation

The previous pages and tables described the scheduling techniques for planning software project management. This section will discuss cost estimation techniques. There are seven basic cost estimation techniques used today. The following descriptions and tables will help show why so many software projects are produced over budget. (Boehm, 1984, p. 242)

a. Algorithmic Models

These models have one formula or algorithm that produces a cost estimate. The algorithm is a function of several variables which are considered important criteria or cost drivers.

b. Expert Judgement

This method involves the judgement of someone in the field of software cost estimation who is considered an expert. The expert alone comes up with the cost estimate. The Delphi technique was developed to prevent the dependency on just one expert. The Delphi technique uses several experts to form a consensus, providing their inputs in a nonattributive way.

c. Analogy

The analogy method compares already completed software development project(s) with the current project. From the comparison of the two software projects a cost estimate is derived for the project not yet developed.

d. Price-to-Win

The price-to-win philosophy states that the software project will cost whatever is necessary to win the contract. The

contract. The idea is to get your foot in the door and be awarded the contract, then make your profit when modifications or changes are done. This method appears to be only viable to software projects that place bids on contracted work. However, this philosophy can apply to non-contracted software, if one can imagine a software project manager stating the cost estimates based on what he/she believes the boss wants to hear.

e. Top-Down

A global price estimate is derived from the entire software project properties first. Next, each of the project's components are estimated from the global price. The component dollar amounts are determined from the ratio each component has in relation to the entire software project.

f. Bottom-Up

The bottom-up method is similar to the top-down approach, except done in reverse. Each software component cost is determined and then summed together for the price of the entire project. Table 4-6 shows the strengths and weaknesses of each of the seven cost-estimation techniques described above: (Boehm, 1984, p. 243)

TABLE 4-6. STRENGTHS AND WEAKNESSES OF COST ESTIMATION METHODS

METHOD	STRENGTHS	WEAKNESSES
Algorithmic model	- Objective, repeatable, analyzable formula - Efficient, good for sensitivity analysis - Objectively calibrated to experience	- Subject inputs - Assessment of exceptional circumstances - Calibrated to past, not future
Expert judgement	- Assessment of representative-ness, interactions, exceptional circumstances	- No better than participants - Biases, incomplete recall
Analogy	- Based on representative experience	-Representative-ness of experience
Price to win	- Often gets the contract	- Generally produces large overruns
Top-down	- System level focus - Efficient	- Less detailed basis - Less stable
Bottom-up	- More detailed basis - More stable - Fosters individual commitment	- May overlook system level costs - Requires more effort

Table 4-6 above shows a general unsatisfactory effort for software cost estimating. One observation that can be made is that no one alternative appears to be better than another. Furthermore, the Parkinson and price-to-win methods (while still used today) do not produce satisfactory software cost estimates. The remaining

methods are balanced between their strengths and weaknesses. It is understandable to see why so many software projects are over budget with the aforementioned cost estimating methods. A combination of many of the methods would need to be performed on a software project and a comparison of the results would need to be made in order to produce a good cost estimation. However, a better software cost estimation method exists. The method is called the Constructive Cost Model (COCOMO).

g. COCOMO

COCOMO's primary concern is to help software project managers understand the cost consequence of decisions made during software development. The COCOMO method provides large amounts of data and forces the software project manager to understand and determine the numerous attributes or cost drivers associated with development. The software tool has three increasingly detailed models. The intermediate level COCOMO will be described. Table 4-7 that follows is used to determine the type of software project. There are three classifications of software projects when using the intermediate COCOMO technique. (1) An organic software mode project is one that comes from a stand alone package or has familiar and stable requirements; (2) The embedded software mode is software that is intrinsically part of the hardware. Therefore, interface specifications for conformance is considerable. The

software may be unfamiliar and have unstable requirements; and (3) The semidetached software mode falls between the other two modes. The software project manager can determine the software mode by matching the features of Table 4-7 with their software project.

TABLE 4-7. CHARACTERISTICS OF THREE MODES OF SOFTWARE

		MODE	
FEATURE	**ORGANIC**	**SEMIDETACHED**	**EMBEDDED**
Organizational understanding of product objectives	Thorough	Considerable	General
Experience in working with related software systems	Extensive	Considerable	Moderate
Need for software conformance with pre-established requirements	Basic	Considerable	Full
Need for software conformance with external interface specifications	Basic	Considerable	Full
Concurrent development of associated new hardware and operational procedures	Some	Moderate	Extensive
Need for innovative data processing architectures, algorithms	Minimal	Some	Considerable
Premium on early completion	Low	Medium	High
Product size range	<50,000 Delivered Source Instructions (KDSI)	<300 KDSI	All sizes

The next step is to estimate the size of the software project. One method is by using historical data from other software projects (example of the analogy method described previously) to determine size of THE software project. The information is then placed into the appropriate formula (example of the algorithmic modeling technique described previously) from Table 4-8 that follows. The formulas under the "NOMINAL EFFORT" column would be used first to determine the Man Months (MM) for the software project. Suppose for example, it was determined that a software project was an organic software mode because the employees were familiar with the software development, the requirements were stable and the estimated size was 20,000 Delivered Source Instructions (20KDSI). The solution would be: $(\text{Man Month})_{NOM} = 3.2(20)^{1.05} = 74$ Man-Months (MM).

TABLE 4-8. FORMULAS FOR THREE MODES OF SOFTWARE

DEVELOP-MENT MODE	NOMINAL EFFORT	SCHEDULE
ORGANIC	$(\text{Man Month})_{NOM} = 3.2(\text{KDSI})^{1.05}$	$TDEV = 2.5(\text{Man Month}_{DEV})^{0.38}$
SEMIDETACHED	$(MM)_{NOM} = 3.0(\text{KDSI})^{1.12}$	$TDEV = 2.5(\text{Man Month}_{DEV})^{0.35}$
EMBEDDED	$(MM)_{NOM} = 2.8(\text{KDSI})^{1.20}$	$TDEV = 2.5(\text{Man Month}_{DEV})^{0.32}$

(KDSI = thousands of delivered source instructions)

This number 74MM is what is referred to as the nominal development effort. The nominal development effort is just that; nominal, which means (cost drivers effecting the project are rated as average or nominal). Some examples of the cost drivers for the intermediate COCOMO technique fall under four general areas of: product attributes (e.g. software reliability, and product

complexity), computer attributes (e.g. execution time constraint, and main storage constraint), personal attributes (e.g. analyst capability, and programmer capability), and project attributed (e.g. use of modern programmer practices, and use of software tools). A total of 15 cost drivers are identified for the intermediate COCOMO technique. It is here the COCOMO technique can work over time for the software project manager. COCOMO can be used for sensitivity analysis. Sensitivity analysis can show the effects on time and money to the software project, by changing one of the 15 cost drivers at a time and reviewing the results. Now, assume our example has all 15 cost drivers categorized as nominal. Then the answer of 74MM represents the number of man months (an estimator for cost) which is the expenditure to produce that software. The final step will show the COCOMO user the actual estimated time to develop the software. Man months are used for cost estimation by themselves or can be converted into actual dollars if so desired. Therefore, this formula "TDEV = 2.5(Man $Month_{DEV})^{0.38}$" from Table 4-8 under the "SCHEDULE" column would be used since our example has previously been determined the organic mode. The solution for how long it would take to develop the 20KDSI organic mode software is: $TDEV = 2.5(74MM)^{0.38} \sim 13$ months (approximately).

Finally, the COCOMO technique has been calibrated by the use of 63 software projects that range from business, scientific, systems, real-time, and support. The technique has estimated the software projects within ± 20 percent of actual costs

approximately 68 percent of the time. Furthermore, depending on the use of the technique, COCOMO can be calibrated to fine tune it to better accuracy. (Boehm, 1984, pp. 248-251)

7. Set Procedures and Rules

> In contrast to policies, procedures establish customary methods of handling future activities and provide guides to action rather than to decision making. Procedures detail the exact manner in which to accomplish an activity and allow very little if any definite actions to be taken or not taken with respect to a situation and allows no discretion." (Thayer, 1988, p. 24)

8. Develop Programs

The program (not the same as a computer program) is the collective items of specific goals, policies, procedures, and rules interacting with the tasks, and resources to achieve a desired course of action. The software project manager pulls together the three areas of; tasks necessary for the completion of the software project, cost and resources needed, and the schedule.

9. Forecast Future Situations

The software project manager would best be suited if he/she had a crystal ball to look into the future. The first type of future events are the availability of resources. The resources of time, money, people and equipment are some examples for future consideration. A good indicator of the future for a software project is its importance, or its perceived importance from the high management. The second type of future event is the software itself. Will the software be able to handle the user's

requirements of today and be able to expand or be modified easily for the future? This phase is key if the software has been marked for reusability.

10. Prepare Budgets

Budgets are the common thread for comparison in most projects of any kind. A common denominator for all resources of manpower, equipment, travel, or office space can be equated to a dollar figure. The budget is the process of placing the dollar value and constraints on the project. The software project manager is responsible for dividing the appropriated funds available to all of the software project areas.

11. Document Project Plans

In all the phases of project software management it is important to record what decisions were made in documents. Documentation is necessary for continuity in the absence of the manager or any other key project member. The document project plan is also critical for preparing the software project for other key documents that will follow. Examples of key documents contained within the document project plan are: quality assurance plans, staffing plans, and configuration management plans. The project plan is the means for the project software manager to interface with outside organizations interacting with the project. The project plan informs the outside organizations of what the project software manager expects in terms of relationship and responsibilities between the two.

C. ORGANIZING

Some of the major issues for software management organizing can be summarized as follows:

- Difficulties in determining best organizational structure for project
- Undefined or unclear responsibilities for project activities
- Conflict between individual staff and software functional organizations (Thayer, 1988, p. 26)

1. Types of Organizations

The organizing phase in software management can help make the project run smoothly. It is an administrative link that lays the foundations for the chain of command and establishes the lines of authority. The three basic organizations existing in software project management are: functional, project, or matrix organizations.

a. Functional Organizations

Functional organizations are best for adopting highly specialized and very skilled software programmers, because the programmers work in their area of expertise. Figure 4-1 depicts what a functional organization employee chart would look.

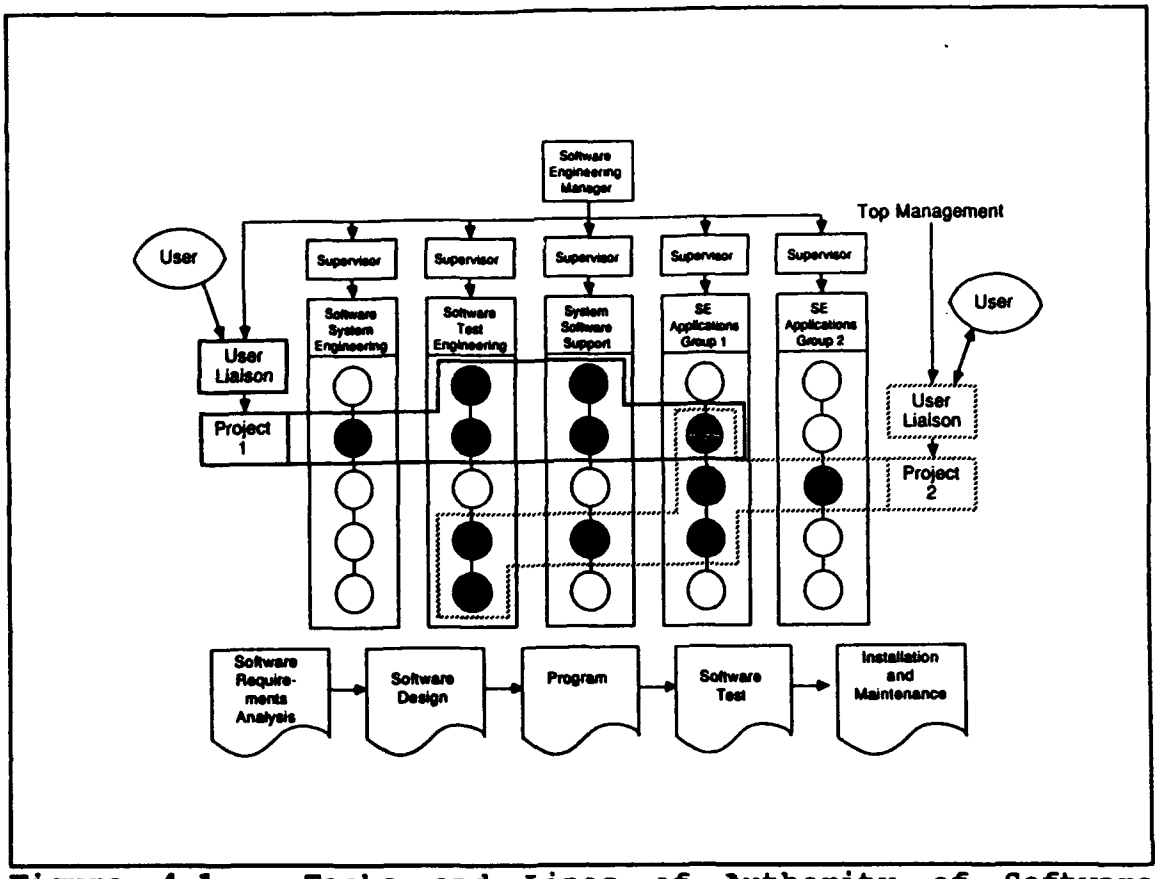

Figure 4-1. Tasks and Lines of Authority of Software Development Functional Organizations Used to Develop a Project (Thayer, 1988, p. 29)

Table 4-9 illustrates the strengths and weaknesses of the functional organization (Thayer, 1988, p. 32).

TABLE 4-9. STRENGTHS AND WEAKNESSES OF THE FUNCTIONAL ORGANIZATION

STRENGTHS	WEAKNESSES
Organization already in existence (quick start-up and phase-down).	No one person has complete responsibility or authority for the project.
Recruiting, training, and retention of people is easier. (functional projects are people-oriented)	Interface problems are difficult to solve.
Standard, techniques, and methods are already established.	Projects are difficult to monitor and control.

b. *Project Organizations*

Project organizations are considered ideal for large software projects, due to the centralized control by one person the software project manager. Figure 4-2 depicts what a project organization employee chart would look.

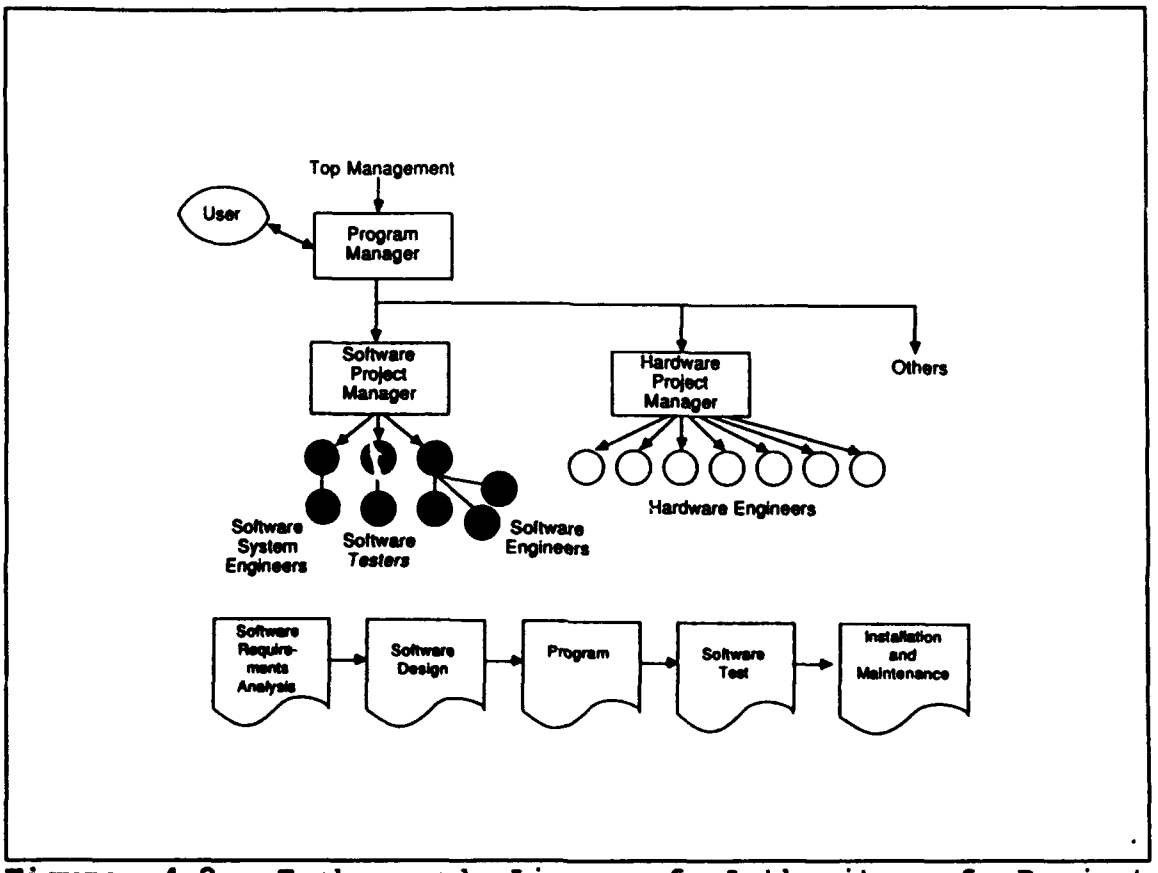

Figure 4-2. Tasks and Lines of Authority of Project Organization Used to Develop a Project (Thayer, 1988, p. 30)

Table 4-10 illustrates the strengths and weaknesses of the project organization (Thayer, 1988, p. 32).

TABLE 4-10. STRENGTHS AND WEAKNESSES OF THE PROJECT ORGANIZATION

STRENGTHS	WEAKNESSES
There is a central position of responsibility and authority for the project.	Organization must be formed.
One person has authority over all system interfaces.	Recruiting, training, and retention of people is more difficult (projects are product-oriented).
Decisions can be made quickly.	Economy of scale cannot be achieved.
Staff motivation is typically high.	Projects tend to perpetuate themselves.
	Standards, techniques, and procedures must be developed (no commonality between projects).

c. Matrix Organizations

Matrix organizations are considered the best compromise between project and functional organizations. Figure 4-3 depicts what a matrix organization employee chart would look.

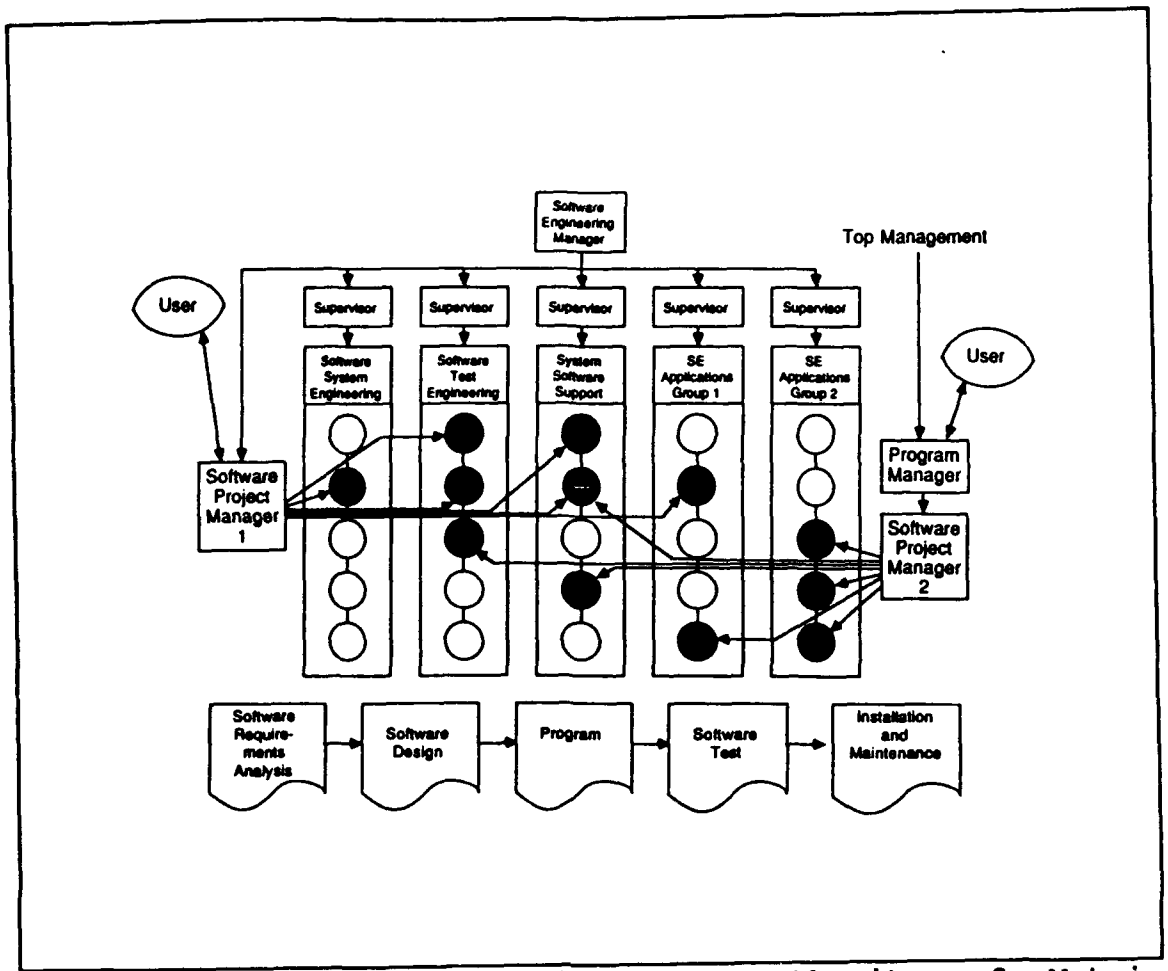

Figure 4-3. Tasks and Lines of Authority of Matrix Organization Used to Develop a Project (Thayer, 1988, p. 31)

Table 4-11 illustrates the strengths and weaknesses of the matrix organization (Thayer, 1988, p. 32).

TABLE 4-11. STRENGTHS AND WEAKNESSES OF THE MATRIX ORGANIZATION

STRENGTHS	WEAKNESSES
Improved central position of responsibility and authority (over functional project).	Responsibility and authority is shared between two or more managers (unlike project organization).
Interfaces between functions can be controlled more easily (than in functional project).	Control or responsibility for resources (people) is shared between two or more managers (unlike project or functional organization).
Recruiting, training, and retention are easier (than in project organization)	Too easy to move people from one organization to another (unlike project or functional organization)
Easier to start and end the project (than in project organization).	Greater organizational documentation is required (than in project or functional organization).
Standards, techniques, and procedures already established (unlike project organization).	Greater competition for resources (than in project or functional organization).
Better and more flexible use of people (unlike project or functional organization).	

Organizing activities and tasks are created to help relieve the difficulties and problems associated with the major issues mentioned earlier. The detailed steps and activities for organizing are:

2. Identify and Group Required Tasks

The software project manager must first identify the tasks that need to be done. This procedure can be done by using a work breakdown structure, where the tasks are identified and then grouped in common functional areas. The groupings can be placed in

areas like software system programming, software testing...etc The software project manager must not only group the tasks from inside the realm of the software project but also outside where interactions occur.

3. Select and Establish Organizational Structures

A selection of the three general types of project organizations identified earlier must be chosen, after the tasks have been identified and grouped by functional or areas of responsibilities. The strengths and weaknesses should be considered as a means of keeping the organizational structure in line with the software project goals and type of project is being developed.

4. Create Organizational Positions

The organizational structure is basically the shell. The positions that go into the shell will add the definition and meat to the organization. Examples of some common types of positions and their job description are as follows:

- Project managers - responsible for system development and implementation within major functional areas. Direct the efforts of software engineers, analysts, programmers, and other project personnel.
- Software engineers - design and develop software to drive computer systems. Develop firmware, drivers, specialized software such as graphics, communications controllers, operating systems and user friendly interfaces. Work closely with hardware engineers and applications and systems programmers, requiring understanding of all aspects of the product.

- Scientific/Engineering programmers, programmer-analysts - perform detailed program design, coding, testing, debugging, documentation and implementation of scientific/engineering computer applications and other applications that are mathematical in nature. May assist in overall system specification and design. (Thayer, 1988, pp. 33, 34)

5. Define Responsibilities and Authority

Defining responsibilities is important because it helps prevent items from dropping through the cracks (meaning no one was doing that task). Responsibility definition should also reduce confusion of which section in the organization is doing what work for whom. The definition of authority is the way to provide lines of authority. Lines of authority are ways of detailing what tasks has precedence over other tasks. An example may be that it is decided that the software analysis phase has authority over the software design phase.

6. Establish Position Qualifications

Once the organizational positions have been created and their job description identified; it is necessary to define the type of qualified person. The following are examples of the previously identified organizational position titles and their job qualifications:

- Project managers - background in successful systems implementation, advanced industrial knowledge, awareness of current computer technology, intimate understanding of user operations and problems, and proven management ability. Minimum requirements are four years of significant system development and project management experience.

- Software engineers - four years experience in aerospace applications designing real-time control systems for embedded computers. Experience with Ada preferred. B.S. in Computer Science, Engineering, or related discipline.

- Scientific/Engineering programmers, programmer-analysts - three years experience in programming aerospace applications, control systems, and/or graphics. One year minimum with FORTRAN, Assembler, of C programming languages. Large-scale or mini/micro hardware exposure and system software programming experience desired. Minimum requirements include undergraduate engineering or math degree. (Thayer, 1988, p. 34)

7. Staff Positions

Staffing positions is a major effort of the organization phase of software project management. Staffing requires the choice of filling the position from within the organization or hiring off the street (outside the organization). Hiring standards must be set and applied. Training and education of employees may be necessary to assimilate them into a productive worker on the software project team. Also, a means for development in the areas of professional knowledge and skills must be planned for after the employee has been hired. Finally, a means of evaluating and appraising personnel is needed in conjunction with a reward or disciplinary actions.

8. Document Organizational Structures

Again the importance of documenting comes up in every phase of the software project management. The type of organization chosen and reasons for the decision should be recorded. Other items worth documenting are the lines of authority, tasks, responsibilities, descriptions, and qualifications for the positions created should be included in an organizational plan.

D. DIRECTING

Some of the major issues for software management directing can be summarized as follows:

- Communications barriers between key organizations involved in the software project
- Lack of motivation or fears of change by trying new methods
- Ways of motivating software development staff have not been put into action (Thayer, 1988, p. 38).

The directing phase in software management is the one-on-one interface with management and the workers. Its the personal touch of leading and motivating people to reach their potential. Proper directing can create a synergism providing more or better quality work from the employees then thought possible. Directing activities and tasks are created to help relieve the difficulties and problems associated with the major issues above. The detailed steps and activities for directing are as follows:

1. Provide Leadership

An effective manager uses leadership in addition to using management tools to be successful. Managers are very important in today's society. Many important issues and values are derived from society. Managers use values from personal experience and expand on them, with the coordination of the company atmosphere.

The management process has responsibilities to society, the company and the subordinate worker. The manager must be loyal to the organization's goals/missions but yet stay within boundaries placed on the organization, by higher authorities, society and the individual needs of the workers. The manager must help the

organization survive and prosper, because if the organization fails, both society and the employees suffer. Managers are the activating element needed to plan the companies objectives and then achieve them on time. The tools needed to obtain objectives for management are found within the components of the management process.

The software project manager must provide leadership and direction to the team. There are two general types of power that the software project manager (the leader) can display. They can be classified as positional power and personal power. As one could guess, positional power is granted by a person's position or job in relation to others (subordinates). Positional power is the type of power all bosses have over subordinates and is inherited by job description. The second type of power (personal) has nothing to do with what job the person has, but instead what the person himself/herself is like. Any person (boss or subordinate) can have personal power. In fact; a conflict my arise when the boss who has positional power has a run in with the subordinate who has personal power. A power struggle can occur between the two and disrupt the harmony of the organization. Therefore, a software project manager (the leader by job description) will have positional power and may have the second type of power (personal). Why is the discussion of power important? It is the acquisition of power that gives the leader the capability to rule or lead the subordinates. The job of the leader is to interpret the plans and requirements and communicate those ideas to the employees to ensure the

obtainment of a common goal and the objectives for the software project. (Newman, Kirby, and Schnee, 1982, pp. 386-390)

Managers have to deal directly with people very often. The management element that deals mostly with people (the number one resource of any manager) is the directing element. A definition of leadership is as follows: the ability of one person to influence the behavior of another toward the accomplishment of certain goals. Competition is so keen between organizations today, and that is what breeds the growth of leadership.

The four elements of leadership are; implementing, directing, communicating, and coordinating. a) Implementing is the element that gets actions started. b) The directing element is concerned with giving orders. This element assists the implementing element to arrive at the desired set of goals. c) Communication is the transfer of information, ideas, understanding, or feelings between people. The leader must communicate his/hers orders clearly, and concisely, so the workers understand and can carry them out. d) Coordination is the element of leadership that pulls everything together. The coordination element makes sure everyone is informed before key actions are implemented and the desired goals completed.

Leadership is a very important aspect of managing (contained within the directing element of management). To be an effective manager, one needs to be able to be a good leader. One must be able to motivate workers through leadership. The manager must be able to communicate clearly and concisely to his/her

employees. To induce job satisfaction and innovation the manager must insure a feeling of confidence and pride into the employee through communication and leadership. The manager must coordinate with key personnel within and outside the organization to ensure organizational goals are achieved. All the elements (implementing, directing, communicating and coordinating) of leadership are important and necessary for today's manager. (Putnam, 1987)

Four general styles of leadership can be identified as a leader who delegates, participates, sells or, tells subordinates what to do. The four styles are used under different conditions. A software project manager can be one or any combination of these four different styles of leading depending on the situation. Therefore the term "situational leadership" (Hersey and Blanchard, 1982, p. 150) is utilized here. One factor of the employee's maturity level should be taken in consideration when a software project manager is examining the proper situational leadership style he/she should use. Figure 4-4 illustrates all pertinent criteria and gives the reader the ability to extract the proper leadership style when he/she knows the maturity level of workers.

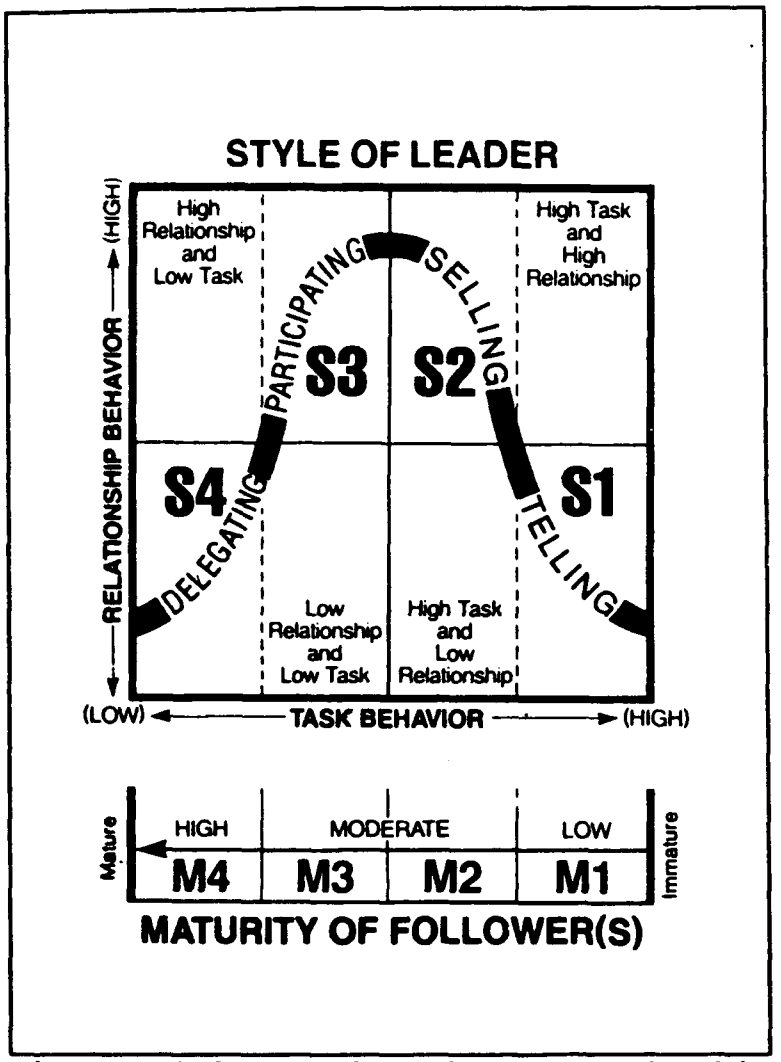

Figure 4-4. Situational Leadership (Hersey and Blanchard, 1982, p. 152)

Figure 4-4 can be a little intimidating, when one first looks at the situational leadership styles, but it is not too complicated. Basically, the figure shows the direct effect for choosing proper leadership style when one considers the employees maturity level. For example the low maturity level employee (M1 in Figure 4-4) requires a response of an S1 leadership style (telling

leader). Figure 4-4 then lets the leader know that a telling leader has to give a lot of tasks to the employee, but needs to have a low relationship level (meaning not explaining the tasks) with the employee. Table 4-12 helps explain when which leadership style should be used for each maturity level. The table is a supplement to ensure the reader understands the situational leadership figure. (Hersey and Blanchard, 1988, p. 154)

TABLE 4-12. MATURITY LEVEL VERSUS LEADERSHIP STYLE

MATURITY LEVEL	APPROPRIATE STYLE
M1 Low Maturity Employee unable and unwilling or insecure	S1 Telling High task and low relationship behavior
M2 Low to Moderate Maturity Employee unable but willing or confident	S2 Selling High task and high relationship behavior
M3 Moderate to High Maturity Employee able but unwilling or insecure	S3 Participating High relationship and low task behavior
M4 High Maturity Employee able/competent and willing/confident	S4 Delegating Low relationship and low task behavior

The four leadership styles of telling, selling, participating and delegating are described in the following paragraphs (Hersey and Blanchard, 1988, p. 154):

a. ***Telling Situational Leadership Style***

The telling leader is dealing with employees of low maturity levels. The low maturity level can be due to the fact of an employee's inability to perform the work at hand or that the employee is insecure. The type of employee that is both unable and

unwilling to do the work is typical for the telling situational leadership style. Therefore, the leader needs to tell the employee what, how, when, and where to do the work. The leader is not able to have a high level of interaction for support of the employee, because this may be seen as rewarding poor performance or the leader is permissive and weak.

b. Selling Situational Leadership Style

The selling leader is dealing with employees of moderate maturity levels. The moderate maturity level is due to the employee being willing to perform the tasks and is confident in his ability, but lack the skills. Therefore, the leader interacts often with these types of employees providing direction and also supportive behavior. In this way he is able to reinforce and give feedback, to allow learning and growth in enthusiasm. A type of two way communications happens between the leader and the follower giving an appearance that the leader is convincing or selling the tasks to the employee.

c. Participating Situational Leadership Style

The participating leader deals with employees who also have a moderate maturity level. However, unlike the selling leader's employees, the participating leader's employees are able but unwilling to do the work. The workers unwillingness to do the work is due to their lack of confidence or insecurity. This type of worker therefore needs motivation (high relationship between the leader and the employee) to get the work done and a two way communication and explanation to see how he/she is doing. The

leader shares in the decision making process with the employee, thus the term participative leadership style.

d. Delegating Situational Leadership Style

The best type of leader is the delegating leader. The employees that deserve this style of leadership are both able and willing to do the work and handle the responsibility. The employee needs little (low relationship between the leader and the employee) help or communications. The employee is told what needs to be accomplished and he/she alone decides how, when, and where to do the task.

2. Supervise Personnel

If leading can be thought of as long term or strategic directing; then supervising can be related to short term or tactical directing. The software project manager must guide and keep track of the employees on a day-to-day basis. Supervision allows the software project manager to assure the employees are doing their assigned duties as well as provide guidance as well as discipline as necessary.

3. Delegate Authority

To free time and effort of the software project manager it may be necessary to delegate the work load and the authority along with it. However, one must realize that delegated authority does not relieve the software project manager (the delegator) from his/hers responsibility. A good manager will delegate authority down to the lowest employee level responsible effort to handle the tasks.

4. Motivate Personnel

An important aspect of directing is to motivate the workers. Motivation can help workers not only get their job done, but to achieve the highest level of performance from them in the process. Table 4-13 illustrates the most common motivational psychologist theories used today and a brief explanation (Thayer, 1988, p. 41):

TABLE 4-13. DEFINITIONS OF MOTIVATIONAL MODELS

MOTIVATION MODEL	DEFINITION OR EXPLANATION
Frederick Taylor	Workers will respond to an incentive wage.
Elton Mayo	Interpersonal (group) values were superior to individual values. Personnel will respond to group pressure.
Kurt Lewin	Group forces can overcome the interest of an individual.
Douglas McGregor	Managers must understand the nature of man in order to be able to motivate him.
A.H. Maslow	Human needs can be classified. Satisfied needs are not motivators.
Frederick Herzberg	A decrease in environment factors is dissatisfying; an increase in environment factors is not satisfying. A decrease in job content factors is not dissatisfying; an increase in job content factors is satisfying.
Chris Argyris	The greater the disparity between company needs and individual needs the greater the dissatisfaction of the employee.
Rensis Likert	Participative management is essential to personal motivation.
Arch Patton	Executives are motivated by the challenge in work, status, the urge to achieve leadership, the lash of competition, fear, and money.
Theory Z	A combination American and Japanese management styles. People need goals and objectives, otherwise they can easily impede their own progress and the progress of their company (Ouchi, 1981).
Quality circles	Employees meet periodically in small groups to develop suggestions for quality and productivity improvements.

Most software engineering personnel are well paid, work in pleasant surroundings, and are reasonably satisfied with their position in life. Therefore, in accordance with Maslow's

> hierarchy of unfulfilled needs, the average software engineer is high on the ladder of satisfied need. Most software engineers are at the 'esteem and recognition' level and are occasionally reaching to the 'self-actualization' level. Thus, one of the issues that management is facing today is the question of what it takes to motivate software engineers into producing more and better software (called software psychology in some circles), since money alone doesn't seem to do it. (Thayer, 1988, p. 39)

5. Resolve Conflicts

The software project manager can't be the expert in all areas of the project, but does have to apply good judgement at times. Software projects will not always go according to plan and will have its share of rough times. The software project manager will need to resolve conflicts between personnel or project decisions during rough times. The manager also has to be aware of any conflicts of himself/herself if power struggles or personality conflicts arise.

6. Manage Changes

Many items can and will change in a software development project. Examples of items that may change through the software life cycle are: people, user's requirements, software design, and the hardware for which the software is being written. It is impossible to stop or prevent all changes so the software project manager must learn to manage changes. A simple plan is written to handle one type of change. The change involving a new software development technology (a change) can be handled by following these steps:

- Explain the risks and benefits of the new technology.
- Provide training for the project team.

- Prototype the technique before it is used.
- Provide technical support throughout the project.
- Listen to the users' concerns and problems.
- Avoid concentrating on the technology at the expense of the project. (Thayer, 1988, p. 43)

7. Document Directing Decisions

The importance of documenting comes up in every phase of the software project management. The type of directing decisions and reasons for the decision should be recorded. Other items worth documenting are all tasks, assignments of authority, and any conflicts with their resolutions.

E. CONTROLLING

Some of the major issues for software management controlling can be summarized as follows:

- Software project control relies to often on budget expenditures for software schedule.
- Standards for software development are not written or not enforced.
- Measures of software quality referred to as software metrics are not used often enough (Thayer, 1988, p. 43).

The controlling phase in software management is the phase to monitor and make corrections to areas of budget and schedule. It is the area where predicted project performance is compared to actual project performance. Any disparities in the estimated and actual performance must be reviewed to determine if any action should be taken either rewarding or disciplinary.

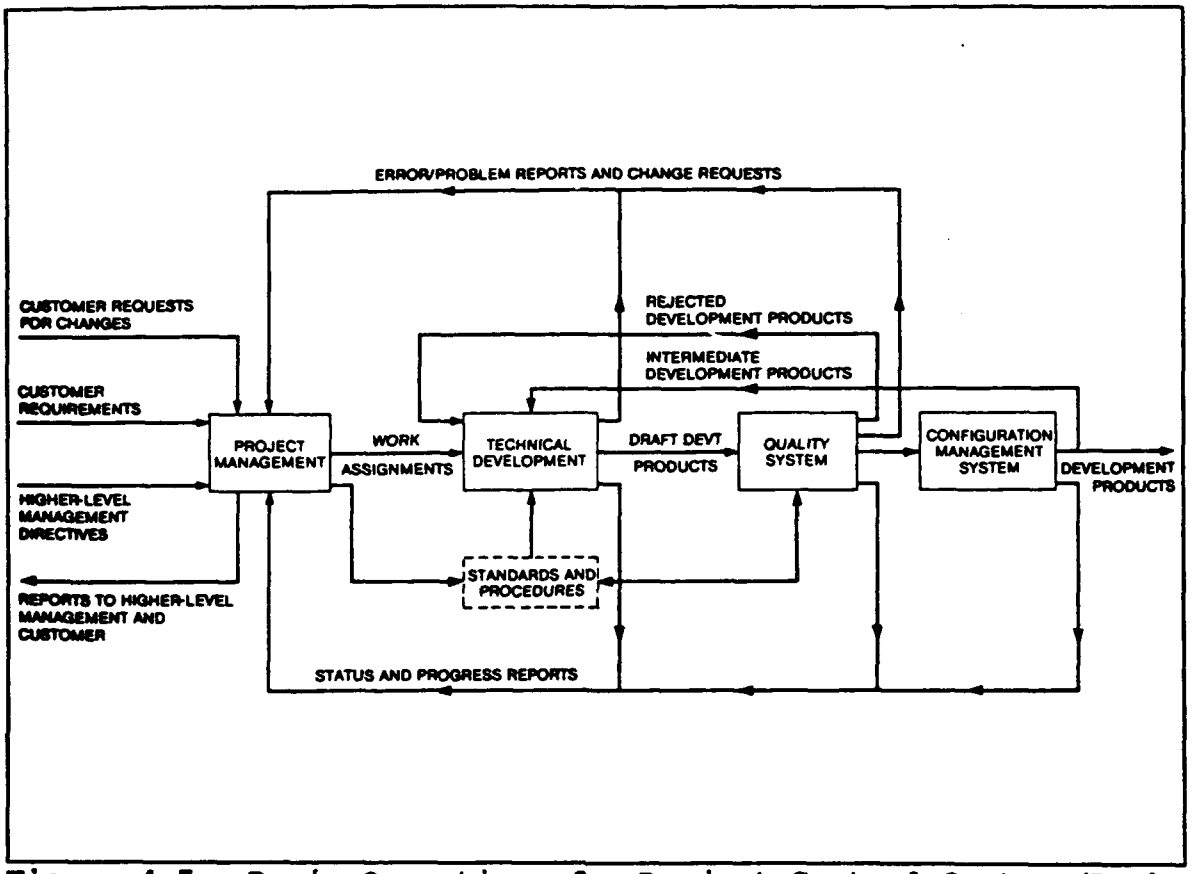

Figure 4-5. Basic Operation of a Project Control System (Rook, 1986, p. 109)

The main points that needs to be made about Figure 4-5 is the interrelationship of project management, and the iterative/feedback method of control. The project management box has five inputs and three outputs when considering the project control system. The inputs on the left side of the box are the taskings or requirements. The three specific items are the original customers requirements, any changes to the requirements (customer requests for changes) and the directives from higher management in terms of policies and procedures. The other two inputs are the interactive/feedback portion from the error/problem reports, and status/progress reports. The feedback is a very important item in

the control process. It lets the software project manager know if he/she has control of the project and the results of any corrections made. Without the iterative/feedback loop the software manager is unable to learn from the results; if corrections actions were determined, or to just monitor the status of the project.

The outputs of the project management box are: reports to higher management, and work assignments. The reports to higher management are the means by which the software project manager provides an interactive/feedback loop. The higher management can see how well or poorly the software project is progressing and determine if involvement is necessary. The other outputs (two work assignments) on the right side of project management box are the processed output of the project management functions of planning, organizing, directing, and controlling (the phase presently in). The processed requirements combined with higher-level management directives produce the work assignments that must adhere to standards and procedures (the dotted line means monitoring involvement unless otherwise needed) during the development cycle. The development software life cycle (system engineering, analysis, design, code, testing, and maintenance) was described in the previous chapter and is contained in the technical development box in Figure 4-5. Finally, to produce the end products (goals, and objectives of the project) and ongoing quality assurance and configuration management control must be applied.

Some of the same tools and techniques for scheduling and cost estimation described in the planning phase of this chapter are used

in the control phase of software project management. A strong and well thought out planning and controlling phases are key to keep costs under budget and the schedule on time. Proper software project controlling entails discovering problems or variations from plans early to allow necessary corrections to be made. Controlling activities and tasks are created to help relieve the difficulties and problems associated with the major issues above. The detailed steps and activities for controlling are (Thayer, 1988, pp. 44-47):

1. **Develop Standards of Performance**

The initial step for controlling is to develop items to measure the project against to monitor the progression of the software project. These items to measure progress are called standards. Standards can be a detailed extension of the goals and objectives in the planning phase of project management. Software standards can be either process or product types. Some types of process attributes that standards are developed for are call quality metrics. Examples of quality metrics are: software reliability, portability, and usability to name a few. The standards developed for products are applied to the software deliverables. Examples of software deliverables are: a feasibility assessment study, a requirements specifications document, and a design plan to name a few. The standards can be developed for the individual software project or may be adopted from the organization's standards or be outside organizational standards (Institute of Electrical and Electronic Engineers (IEEE) for example).

2. Establish Monitoring Techniques and Reporting Systems

Monitoring progress of the software project is the next step once the standards are developed or adopted. Monitoring techniques provide feedback to the software project manager which indicates the need for corrective actions or not. Also the project manager must have a way to receive the information from the monitoring techniques to review and inform the progress of the project up the chain of command. Establishing the monitoring techniques and reporting systems are the foundations for the next two steps of measuring the results and taking corrective actions in necessary. Types of monitoring techniques are the same ones as identified in the planning phase for project management tools and are as follows: milestone chart, full wall scheduling, PERT, Gantt chart, CPM, COCOMO. One of the benefits of these techniques are to allow the project manager to use during various phases of the project development. Examples of the types of reports that would be used in the software project are given in Table 4-14:

TABLE 4-14. TYPES OF SOFTWARE PROJECT REPORTS

TYPE OF REPORTS	DEFINITION OR EXPLANATION
Budget	Compare budget with expenditures and provide for making new budget estimates.
Schedule	Provide status of schedule and milestones completed.
Man-hour by activity	Provide a report on the number of staff hours that nave been worked on a given activity.
Man-day by task reports	Provide the number of days assigned to a given task. A task may or may not be larger than an activity.
Milestone due or overdue	Provide a status of the milestones that have been accomplished or have been missed and the reason for missing the milestone.
Project progress	A free-flowing narrative report indicating the status of progress or a list of activities accomplished.
Activity reports	Provide, typically over a period of time, what activities have been accomplished.
Trend charts	Show trends in such areas as budget, number of errors found in the system, manhours of sick leave, and so on. Trend reports are used to predict the future.
Significant change	A general change report indicating exceptions to the plan and significant changes both good and bad. However, it normally shows a loss of progress.

3. Measure Results

The software project measurements can be made and compared to the desired specifications or level of work. The measurements are made to either the process by using the project management tools (PERT or CPM) or the product. Examples of

techniques to measure products of software development are: unit development folders, walkthroughs or independent auditing. The definitions follow:

- Unit Development Folders (UDF) - The UDF is a notebook kept by the project manager containing items specific to the software development of the project. The purpose of the UDF is to provide an orderly fashion of development of a specific unit or portion of the software product (e.g. design specifications, preliminary design, and code).

- Walkthroughs - A walkthrough is an informal review of software products (same type as an UDF) conducted by co-workers. The purpose is to have an extra set of eyes or inputs on what another person or group is doing and check any potential problems early.

- Independent auditing - More formal than a walkthrough, because it is done by an outside agency. The purpose is to determine is the software is being developed in compliance with requirements, policy and plans (Thayer, 1988, pp. 48, 49).

4. Initiate Corrective Actions

The project manager will decide if any corrective actions are needed, once the measurements are made from the previous phase. Once corrective action is taken to relieve the disparity from the desired output; the important item then will be to again monitor the results (feedback) to see if it worked. This in an iterative process until the final output is satisfactory.

5. Reward and Discipline

It is important to reward or discipline as part of the corrective process. This is the workers way of receiving feedback. A worker will be unable to learn if he/she did a good or bad job without feedback. A general rule is to reward in public and

discipline in private. A reward can be as simple as praise and a pat on the back and discipline could be acknowledging the product is not satisfactory and improvement should be made.

6. Document Controlling Methods

Documentation may be useful in this phase to produce a "case book" way of developing software in successful projects. The type of control techniques chosen and reasons for the decision should be recorded. Other items worth documenting are the progression of the project, problems and the corrective actions taken.

V. STATE OF THE ART SOFTWARE DEVELOPMENT TECHNIQUES

A. INTRODUCTION

The two main topics of research in this thesis are software life cycle development (secondary subject) and management of software development (primary subject). Many techniques, methods and procedures describing how to develop and manage software have been presented thus far. This chapter will address more advanced software life cycle and management techniques to accompany and supplement those already discussed.

B. IMPROVEMENTS TO THE SOFTWARE LIFE CYCLE DEVELOPMENT

A key issue for software life cycle development pertains to the effectiveness and usefulness of the deliverable end items of each phase. The main phases of software life cycle development are system engineering, analysis, design, code, testing and maintenance. Each phase produces an end product needed by the next phase. Unfortunately some end products are not of high enough quality, or do not satisfy the requirements to help do the next phase. The following techniques and technologies are illustrated to help improve the effectiveness and usefulness during the software life cycle development process. (Pressman, 1989, pp. 13-16)

1. Computer Aided Software Engineering

Computer systems can be used to aid in the development of software, as the name Computer Aided Software Engineering (CASE) indicates. Software engineering consists of: software development, project management, software metrics, and software maintenance. A further explanation of software engineering will follow in this chapter. The key item to understand is that CASE can help the software project manager throughout the life cycle and management of a software project. CASE consists of five components: diagramming tools, a centralized information repository, interface generators, code generators, and project software management tools. The following is an explanation of the five components: (Senn, 1989, pp. 260-264)

a. Diagramming Tools

Diagramming tools support and document the analysis phase of the software life cycle. The tools produce data flow diagrams, and program structure charts. The main advantage is the ease in changing the diagrams, charts, and documentation. CASE takes the tedious and undesirable activities away from the software developer during this phase.

b. Centralized Information Repository

The centralized information repository saves all system information data from the software life cycle phases supported by CASE. The information is also known as the data dictionary. The dictionary contains details of system components and the volume and frequency of each activity. The volumes and

frequencies can aid in software project management by analyzing where resources are being used most. The repository also has controls and safeguards to protect access and the integrity of the information.

c. Interface Generators

The interface generators produce interfaces that may be required by the end user to operate the software. The generators also double as the vehicle to allow the software developer users to employ the CASE tools. The generator can produce prototypes of end user interfaces. Some examples of interface prototypes are system menus, and screen presentations.

d. Code Generators

Code generators are the nucleus of CASE. Executable code can be generated directly from the system specifications portion of CASE. The nice feature of this component is the storing of the code once it is developed in the centralize information repository. The stored software can now be a candidate for reuse. Furthermore, the system specifications are also stored if any changes need to made. Approximately seventy-five percent of a software project's code can be generated from CASE. Therefore, the remaining twenty-five percent of the code still needs to be hand written.

e. Project Software Management Tools

The previous four CASE components support the software life cycle directly. This last CASE component supports the management of software development. The software management

tools support the project manager by providing efficient and effective assistance throughout the software development process. The tool provides support in schedule monitoring to allow the software manager proper use of the available resources. Finally, the CASE management tool allows the software project manager the capability of analyzing specific areas for trends and control feedback. The software manager can personally select specifics of people, department, or processes for example.

f. Integrating the Five CASE Components

The five components of CASE will not be effective if not joined in a manageable fashion producing a whole system. The consolidation of the CASE components are done by the integrating tools. The integration can occur in three of the following ways.

- Creating uniform interfaces
- Providing transferable data upon the different components
- Linking the activities

CASE can be a very useful tool to the software project manager. CASE efficiently produces documentation and code. However, some issues need to be considered before CASE is used for software development and project management. Table 5-0 illustrates some of the strengths and weaknesses associated with CASE. (Senn, 1989, pp. 277 - 280]

TABLE 5-0. STRENGTHS AND WEAKNESSES OF CASE

CASE STRENGTHS	CASE WEAKNESSES
Ease of revising system description and graphic representation	Relies on structured traditional software life cycle development methodologies
Facilitates system prototyping	Absence of standards for methodology support
Generates code	Lack of standards for diagram representation
Provides maintenance support through storage in centralized database	Limited function - primarily supports one specific phase or method of software development life cycle
Increases probability of meeting users requirements	Limited scope - virtually no analysis of applications of requirements

2. Software Prototyping

a. *Definition of Prototyping*

A definition of prototyping can be found, if one looks at the Greek derivation of the word "protos" which means first and "tupos" which means model. Prototyping is the first attempt to make a working model (usually scaled down in size and complexity) of the system. Department of Defense software development projects have written code on the premise of completing the definition of all software functions first. However logical this practice may sound, it does not account for the following illogical reasoning. First, the English language is not a great way of explaining technical software statements, especially when accomplished by humans. Second, one can not forecast all the

problems that will be encountered when writing code. Finally, it seems that software requirements are always in a state of change. Therefore, it is necessary to consider the possibility of creating code at a much early stage and in fact throughout the total software development cycle.

b. Rapid Prototyping

A version of prototyping which produces workable models early and continuously throughout the software development process is called rapid prototyping. Rapid prototyping accentuates the ongoing development of system prototypes that meet the system designers and end users requirements and provide appropriate cost-performance trade-offs. Rapid prototyping requires heavy amounts of software and hardware support, due to the speed of developing models. Furthermore, this process requires continuous feedback between the developer and the end user to increase the flow of ideas which help reduce cost by improving performance or solving problems. Figure 5-0 shows how rapid prototyping requires development work in all phases of software development.

Military and civilian software is increasing in size and complexity. Software development will take an enormous amount of man-hours to accomplish in the future. The normal software development approach of sequentially coding, compiling, integrating and testing one phase at a time will have devastating effects on the schedule. Rapid prototyping or something similar will be necessary to produce the amounts of code required. This continuous development during ongoing stages of the software development will

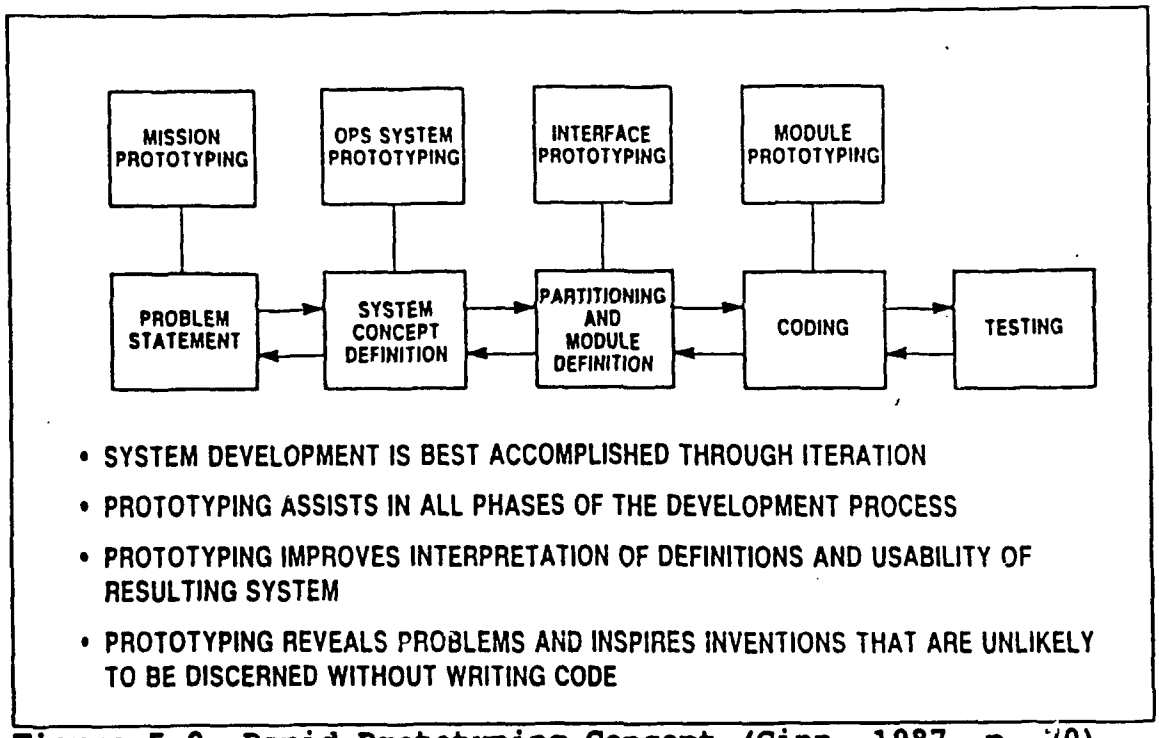

Figure 5-0. Rapid Prototyping Concept (Ginn, 1987, p. 70)

help avert the programmer from writing incorrect code. Rapid prototyping will provide an immediate acknowledgement that the statement is in error. The speed and efficiency of producing software increases when rapid prototyping is combined with graphical interface tools to allow a true user friendly working environment. Rapid prototyping is designed to be an iterative process being able to cope with change and levelop software to respond quickly and efficiently. Also, rapid prototyping techniques can be used to help catch defects/errors early in the software development cycle. Finally, the technique can generate the software for upgrading and modifying the system continually through its life cycle. (Ginn, 1987, pp. 68-70]

C. IMPROVEMENTS TO THE MANAGEMENT OF SOFTWARE DEVELOPMENT

The management of software development needs to have a broad systems approach. The manager needs to beware of all important aspects of the software project, which not only include the development, but is quality, usability, reliability, and risks involved throughout the software's life cycle. The following sections help pull together the management aspects of software development.

1. Software Engineering Approach

The software engineering approach is akin to a systems approach for software. Software engineering can be defined as:

> The practical application of computer science, management, and other sciences to the analysis, design, construction, and maintenance of software and the documentation necessary to use, operate, and maintain the delivered software system. (Thayer, 1988, p. 55)

The approach pulls together four main areas to give a synergistic effect. The four areas are: software development, project management, software metrics, and software maintenance. Each of these four areas are reviewed in the following sections.

a. Software Development

The Software life cycle was discussed in length in chapter III. It consists of system engineering, analysis, design, code, testing, and maintenance phases. The software development procedure is similar except it does not have the maintenance phase included. The software development portion of software engineering actually produces the deliverables (end products of code, and

documentation). In software engineering software development would be on the same level functionally as the other three areas (project management, software metrics, and software maintenance).

b. Project Management

Project management was also discussed in great length in chapter IV. Project management is composed of four elements: planning, organizing, directing, and controlling. The project management portion of software engineering provides the tools and techniques to complete the software project on time and under budget.

c. Software Metrics

Software metrics is a new research topic in this thesis. Software metrics are measurements of certain properties of the software's functionality, and physical aspects. Software metrics can be used as a means to both measure the quality and to control the productivity of software projects. Software metrics can be used in various stages of software engineering. The four classifications of software metrics are: development productivity, project management, quality, and software development.

Three of the four metric classifications (development productivity, project management, and software development) deal with controlling the software project. The fourth metric classification (quality) works directly with improving software quality. The development productivity metric controls a project's cost and manpower; the project management metric tracks the progress of the software project; and the

software development metric monitors the software in the development cycle.

Defects per 1,000 lines of code is what most people would think is a good indicator of software quality. However, this creates a software paradox. A paradox is a contradiction in terms. Take for example the comparison of a software development of functionally exact projects in three different programming languages of APL, PL/I, and Assembler. The delivered programs would all have the same functions but the sizes would vary because APL is the highest order language (of the three) then PL/I and last is Assembler. The higher level languages need less lines of source code to produce, because more computer procedures can be given by one line code compared to lower level languages. Table 5-1 shows the relationship of lines of code needed in each language. The table shows that defects would be the same in all areas of the software life cycle except for the coding phase. The difference in defects in the coding phase is due to the difference in sizes of the developed source code.

TABLE 5-1. COMPARISON OF DEFECTS PER 1,000 LINES OF CODE

	ASSEMBLER	PL/I	APL
DEFECT SOURCE			
Requirements	500	500	500
Design	1,500	1,500	1,500
Coding	4,000	1,000	500
Documentation	1,000	1,000	1,000
Total defects	7,000	4,000	3,500
Source lines	100,000	25,000	10,000
Defects per 1,000 lines of code	70	160	350

The paradox occurs in this example of defects per 1,000 lines of source code, because the higher language software (APL) may be considered a lower quality then the other two software. Furthermore, the large difference in defects per 1,000 lines of code could lead one to incorrectly believe quality of the Assembler language program was higher. Therefore improvements are needed in measuring quality in software as well as in controlling software projects. (Jones, 1986, p. 9)

Examples of the software quality metric factors include the software's: flexibility, testability, reliability, and reusability. The steps for using software metrics can be summarized in the following manner. First, decide on the factors that are important to the software project and its development. Next, determine the criteria in which the measurements are to be compared to. Then establish a way to measure the metrics, and finally compare the two and see if any actions are needed. (Ramamoorthy, Prakash, Tsai, and Usuda, p. 67)

d. Software Maintenance

Software maintenance, briefly discussed in chapter III, encompasses more than just debugging or fixing errors in software. Software maintenance needs to handle the modifications and revision upgrades. Furthermore, the maintenance component must provide for user request changes for the remaining life of the software.

2. Risk Management

One way to improve the management of software development is to reduce the risks involved. A software project manager would be very content if he/she knew the critical risks in their project and could plan to either eliminate, avoid, or at least reduce the effects. The successful software project managers are good risk managers.

Two primary steps of risk management are risk assessment and risk control. Risk assessment involves knowing what the risks are; understanding the effect on the software project; and ranking the risk in order. Risk control deals with planning how to handle the risks; taking action to prevent risks from occurring; and monitoring the results. Software development projects have three basic risk factors to contend with. The three factors of size, structure, and technology. Three basic principles can be derived from the basic risk factors. (1) The larger a software project is; the higher the risk. (2) The higher the structure (well defined) a software project is; the lower the risk. (3) The higher the technology (more complex) that is supported by the software; the higher the risk. The following is a step by step method to develop a risk management plan. (Boehm, 1989, pp. 1-4)

a. Risk identification

A software manager must first be able to identify all the possible major risks involved in their line of work. Several ways of identifying risk for software development projects exist. A simple, but yet effective way is checklists. The

software project manager uses checklists of common software risks to see if they apply to his/her project. An example of the ten most common software risk items and their appropriate technique to control them are as follows in Table 5-2: (Boehm, 1989, p. 8)

TABLE 5-2. PROJECT RISK ITEMS AND SOLUTION TECHNIQUES

RISK ITEM	RISK MANAGEMENT TECHNIQUES
1. Personnel shortfalls	-Staffing with top talent; job matching; team building; morale building; cross-training; pre-scheduling key people
2. Unrealistic schedules and budgets	-Detailed multi-source cost & schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	-Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals
4. Developing the wrong user interface	-Prototyping; scenarios; task analysis
5. Gold plating	-Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirements changes	-High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	-Inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	-Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; team building
9. Real-time performance shortfalls	-Simulation; modeling; prototyping; instrumentation; tuning
10. Straining computer science capabilities	-Technical analysis; cost-benefits analysis; prototyping; reference checking

b. Risk Analysis

Risk analysis involves looking closely at all the risks identified for your software project. Two types of techniques to do risk analysis are with models or analysis methods. Examples or both are: performance and cost type models; and network, decision, and quality factor analysis methods.

c. Risk Prioritization

The last phase in the primary risk assessment step and involves ranking the risk in order of highest (most critical) to lowest. The prioritization phase compares the results from the analysis phase. One way of determine the prioritization of the project risks is using the risk exposure method. This method is basically finding the expected value of the possible risks using the probability of risk occurrence multiplied the dollar amount for lost. Finally, the expected values for each risk would be ranked.

d. Risk Management Planning

Risk management planning is the first phase in the primary risk control step and involves planning to control the risks. Some techniques for risk management planning are risk reduction, risk transfer, and risk avoidance.

e. Risk Resolution

Risk resolution put the risk management plan in action. Two methods for this phase are simulations or prototypes. Prototyping, discussed earlier in this chapter, involves applying the risk plan on a scaled down version of the software development project. Simulations are imitations or models that represent the

real world situation or problem. Therefore, the risk management plan is applied to either the simulation or the prototype to determine the effects of risk.

f. Risk Monitoring

The results of the risk management plan are monitored in this phase. One method of risk monitoring is milestone tracking. The manager observes the actual progress of a software project and compares it to the risk management plan. Next, reassessment of risk must be accomplished to identify and analysis the risks remaining. The process is repeated, thus corrective actions will be necessary if risks are identified.

3. New Software Acquisition Methodology For Military

The United States Department of Defense is the largest user of computers and computer software in the world. The military has needs of normal civilian type software for payrolls and inventories to advanced software for real-time weapon systems, and command, control, and communication systems. The military software life cycle development was discussed in chapter III of the thesis.

The military software development follows the classical software life cycle development. However, the military breaks down each of the phases into smaller well defined and more strictly controlled sections. Another main difference between the military and civilian software is the degree of documentation. The military has many specifications (Military Specifications (MilSpecs)) that require large amounts of documentation. (Jones, 1986, pp. 124-126)

Methodical procedures for software development are established for military software in regulation DOD-STD-2167A (as discussed in chapter III). The regulation requires numerous deliverable products, reviews, audits, and baselines. This process basically reduces flexibility for software development by freezing requirements at each stage. A software contractor can't go on to the next phase until the previous one is completed. However, the previous phases of software development are, modified, or added/deleted if requirements change.

> Development of software for large-scale real-time systems generally has been a failure-prone activity. This tendency can be traced to both lack of control and lack of flexibility. Failure to follow a rigorous software engineering process results in a lack of control because government and contractor management are uncertain that the software effort is being directed toward the correct problem. (Ginn, 1987, p. 76)

Therefore a methodology to allow more flexibility in military software acquisition needs to occur. Other approaches or supplements to DOD-STD-2167A for defense system software development are being pursued to provide more flexibility in software development. The following are examples: 1) Software prototyping: Discussed previously in this chapter. 2) Risk-driven approach: This approach takes the risk management plan, which is also described in this chapter, and places it into effect. The idea is to develop software with the least amount of risk involved. The types of risks and the criteria developed are dependent on the software project manager or by a future regulation for the Department of Defense.

These two approaches will be able to provide flexibility in the military software development cycle. However, it has not been determined how the approaches might be joined together or with the DOD-STD-2167A regulation. Furthermore, the other techniques and methods covered in this chapter may lend credence that improvements for future military software developments are possible. (Ginn, 1987, pp. 76-78)

VI. SUMMARY AND CONCLUSIONS/RECOMMENDATIONS

A. SUMMARY

The primary objective of this thesis was to aid program managers dealing with the development and management of computer software for the military. The thesis was written for program managers who aren't computer experts or don't have much experience in the field. Therefore, the thesis will act as a lessons learned and a how to paper.

Chapter I described the history and importance of the computer and its software. Chapter II examined inherent common difficulties with software development for civilian and military projects. Cost over-runs and late schedules were the two main concerns researched. Chapter III described the classical software life cycle. The life cycle consisted of system engineering, analysis design, coding, testing and maintenance. Chapter IV covered software project management consisting of: planning, organizing, directing, and controlling. Management of software development was the primary theme of the thesis. Various comparisons and tables were made showing how to select the proper scheduling technique. Several cost estimating methods were also discussed. Chapter V gave examples of state of the art methods to aid the software project manager. Two general types of methods researched were areas to support software development and project management.

B. CONCLUSIONS/RECOMMENDATIONS

1. General

- Problems exist in developing software under budget and on time.
- Software development and management have many associated difficulties and risks involved.
- Many scheduling and cost estimating techniques exist, due to great variations in software projects.
- No one scheduling technique is best for all varying software projects.
- Proper aids in software development and software management can help reduce cost over-runs and late projects.
- Use of state of the art methods can aid the software project managers in software development and management.
- Proper software development, effective project management and the methods discussed should help reduce the numbers of cost over-runs and late software projects.

2. Primary Importance of Software Development

- System engineering and analysis design are the most important aspects of software development components (system engineering, analysis design, code, and test)

a. Keys to System Engineering and Analysis Design

- An understanding of the entire system and the relationship of its parts.
- Proper definition and decomposition of the system.
- Good documenting of diagrams and charts to explaining the system will aid phases of coding, and testing.

3. Primary Importance of Software Management

- Planning and controlling are the most important aspects of software project management components (planning, organizing, directing, and controlling).

a. Keys to Planning

- The examination of the possible scheduling techniques to tailor fit your software project is necessary.

- Use table 4-5 (SELECTING A SCHEDULING TECHNIQUE) in chapter IV as a guide. The table listed most of the project scheduling techniques with criteria to help aid the project manager choose.

- Intermediate COCOMO was presented for cost estimating, but different COCOMO models exist for varying amounts of detailed desired.

- COCOMO can be used for sensitivity analysis. Sensitivity analysis can show the effects on time and money to the software project, by changing one of the 15 cost drivers at a time and reviewing the results.

- Risk management should be a key concern to the software project manager. Table 5-2 (PROJECT RISK ITEMS AND SOLUTION TECHNIQUES) in chapter V should be used when developing a risk management plan.

b. Keys to Controlling

- Good planning is essential to controlling. Planning techniques for scheduling, cost estimating and risk managing apply in the controlling component of software management.

- Metrics are very productive in aiding the software manager's goal of produce quality software on time, within budget, and satisfying requirements.

- Be aware that controlling is an iterative process were monitoring and feedback apply. The manager must examine the results of corrective actions to determine if more are necessary.

4. Overlooked Aspect of Software Management

- Directing is often perceived as unimportant, due to the image of low returns from effort exerted. Many project managers may get heavily involved in planning and controlling the project, while not motivating or leading the workers.

- Worker dissention or lack of motivation in some cases can be as detrimental as poor planning and control.

a. Keys to Directing

- The key to directing is the use of the proper situational leadership style (telling, selling, participating, and delegating).
- Use figure 4-4 (Situational Leadership) and table 4-12 (MATURITY LEVEL VERSUS LEADERSHIP STYLE) in chapter IV as reference material to aid in selection of leadership styles.
- Motivating people is essential for top performance from them.
- Use table 4-13 (DEFINITIONS OF MOTIVATIONAL MODELS) in chapter IV as a guide to choose the best motivation model for any situation.

5. Checklist to Aid Software Management

The following is a checklist design to help the software project manager. The checklist will aid in getting started in the management of the project, by forcing the manager to answer the critical questions or addressing the areas in the four components of management.

Planning

1) What are the software project's objectives and goals?

2) What are the organization or top management strategies for achieving project objectives and goals?

3) Do the project's objectives and goals match or are in concert with the organization's? If not, a realignment of project objectives and goals may be necessary.

4) What are the more detailed policies of the organization for software development and project management? Can they be adopted to accomplish your software project? If not, then development of new policies may be necessary.

5) What are the courses of action that can be taken to accomplish your software project?

6) Which is the best course of action to take?

7) What are the organization's procedures and rules for software development? Can they be adopted to accomplish your software project? If not, then new procedures and rules may have to be created.

8) What resources do you need? What resources do you have? What are the risks for the software project? Develop programs to use and measure resources and avoid risks.

9) What does the future environment look like in terms of your software project and the use of its resources?

10) What is the budget for your software project? One needs to be prepared.

11) Document the work done in the planning phase.

Organizing

1) What are the tasks to accomplish the software project? Group tasks into common areas (partitioning)

2) What are the organizational structures needed to accomplish your software project? Identify the staff organizational, project, and team structures. Identify and establish the relationships between project users and developers.

3) What are the duties and relationships for the organizational structures?

4) What are the responsibilities and authorities of the organizational structures?

5) What are the qualifications needed to fill the positions established by the organizational structures?

6) Document the work done in the organizing phase.

Directing

1) What type of leadership style are you more closely related to? What are your strengths and weakness as a leader? Provide leadership by matching personnel goals to project goals.

2) What type of day to day instructions are needed? Provide the determined type of supervision.

3) What work can be delegated? Delegate to the lowest employee level possible that can handle the work.

4) Identify how you can motivate your workers to perform to their highest levels? Motivate your workers.

5) What work and efforts needs to be coordinated and by whom? Coordinate with the appropriate people and staff.

6) Identify any conflicts between project personnel and outside sections. Resolve conflicts.

7) What can be done to allow innovation and independent thought to improve software management? Allow and review innovative thoughts.

8) Document the work done in the directing phase.

Controlling

1) What are your standards of performance for your software project (including software quality assurance methods)?

2) What methods or software tools are preferred in monitoring progress of the project?

3) What items should be measured to best monitor progress? Measure those items.

4) Are the measured items behind planned projections? If so, corrective action are necessary and discipline actions maybe required. If not, corrective action are not necessary and reward actions maybe required.

5) Document the work done in the controlling phase.

LIST OF REFERENCES

Abdel-Hamid, Tarek, "*Software Engineering and Management*," Monterey, California, 1990. Lecture presented at the Naval Postgraduate School.

Boehm, Barry W., "*Software Risk Management: Principles and Practices (Video Notes)*," IEEE Computer Society Press, 1989.

Boehm, Barry W., "*Software Engineering Economics*," IEEE Computer Society Press 1984.

Congress of the United States, Office of Technology Assessment, *SDI Technology Survivability and Software*, May 1988.

DOD-STD-2167A (Department of Defense), Military Standard: Defense System Software Development, 29 February, 1988.

Donohue, Kent A. Cori, and Associates Inc., "*Fundamentals of Master Scheduling for the Project Manager*," Project Management Journal, June 1985.

Ginn, Terry, Chairman of "*AFCEA SDI BM/C^3 Technology Study," Phase I*, August 1987.

Hersey, Paul and Blanchard, Kenneth H., *Management of Organizational Behavior: Utilizing Human Resources*, Prentice-Hall Inc, 1982.

Jones, Capers, *Programming Productivity*, McGraw-Hill Book Company, 1986.

Jones, Carl R., "*C^3 Systems: Structure, Process and Dynamics*," Monterey, California, 1990. Lecture presented at the Naval Postgraduate School.

Kitfield, James, "*Is Software DOD's Achilles' Heel?*," Military Forum, July 1989.

Markland, Robert E., and Sweigart, James R., *Quantitative Methods: Applications to Managerial Decision Making*, John Wiley & Sons, 1987.

Newman, William H., Warren, E. Kirby, and Schnee, Jerome E., *The Process of Management 5th Edition*, Prentice-Hall, Inc, 1982.

Pressman, *Roger S., Software Engineering: A Beginner's Guide*, McGraw-Hill Company, 1989.

Putnam, Lawrence E., *"Engineering Administration,"* Hampton, Virginia, 1987. Lecture presented at the Langley-NASA facilities.

Ramamoorthy, C.V., Prakash, A., Tsai, W., Usuda Y., *"Software Engineering: Problems and Perspectives,"* IEEE Computer Society Press, 1984.

Report of the Defense Science Board Task Force on Military Software, Office of the Under Secretary of Defense for Acquisition, September 1987.

Rook, Paul, *"Controlling Software Projects," Software Engineering Journal*, January, 1986.

Schlender, Brenton R., *"How to Break the Software Logjam," Fortune Magazine*, September 25, 1989.

Senn, James A., *Analysis & Design of Information Systems*, McGraw-Hill, 1989.

Stevens, Louis D., *"Computing Devices and Systems,"* Monterey, California, 1989. Lecture presented at the Naval Postgraduate School.

Thayer, Richard H., *"Software Engineering Project Management a Top-down View,"* Computer Society Press of the IEEE, 1988.

Weik, Martin H., *Communications Standard Dictionary*, Von Norstrand Reinhold Company Inc, 1983.

Witten, Bentley, and Barlow, *System Analysis and Design Methods*, Irwin 1989.

Wulforst, Harry, *Breakthrough to the Computer Age*, Library of Congress Cataloging in Publication Data, 1982.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	AFIT/CIRK Wright-Patterson AFB, Ohio 45433-6583	2
3.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
4.	Prof. Tarek Abdel-Hamid Code AS/Ha Naval Postgraduate School Monterey, California 93943	2
5.	Prof. Donald A. Lacer 5621 Sunmist Drive Rancho Palos Verdes, California 90274	2
6.	C^3 Academic Group, Code CC Naval Postgraduate School Monterey, California 93943	2
7.	Capt Samuel M. Liberto c/o Capt. Luis M. Tirado, Jr. 5024 Rain Drop Circle South Colorado Springs, Colorado 80917	2